Together® Edition for JBuilder®
User Guide

VERSION 6.1

Borland®
Together® Edition for JBuilder®

Borland Software Corporation
100 Enterprise Way, Scotts Valley, CA 95066-3249
www.borland.com

Borland Software Corporation may have patents and/or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents.

Copyright © 2003 TogetherSoft Corporation, a wholly owned subsidiary of Borland Software Corporation. All rights reserved. All patents pending. All Borland brand and product names are trademarks or registered trademarks of Borland Software Corporation in the United States and other countries. Other product names are trademarks or registered trademarks of their respective owners.

Updated May 15, 2003

Contents

Preface 15

Audience 15
Together Documentation Set 15
Finding out more 16
Third-Party Books 16
On the Web 17

Getting Started 19

Tour of Together 21

Getting started with a sample project 21
Opening the CashSales project in JBuilder . . 22
Opening a project. 22
Opening a diagram. 23
Observing correlation between model and code 23
Activating special features 24
Navigating the Together main window 25
Arranging panes and the Inspector 25
View or hide a pane 25
Resizing or expanding panes to full screen 26
Undocking and moving panes 27
Hiding pane title bars 27
The main menu, toolbar, and status bar 28
Main menu 28
Main toolbar. 29
Status bar 29
Explorer pane 30
Directory tab 30
Server tab 31
Model tab 31
Opening diagrams 32
Model, Favorites, and Diagram tab toolbar 33
Favorites tab 33
Diagrams tab 34
Modules tab 35
Components tab 36
Explorer pane operations 36
Right-click menus. 37
Quick project access 37
Copying and pasting 37
Finding the locations of nodes, files, and folders 38
Designer pane 38
Opening and closing diagrams 38
Designer pane toolbar 38
Designer tools. 39
Message pane 39
Property Inspectors 40
Overview of Inspectors 41
Inspector tabset 42
Properties tab. 43
Hyperlink tab 43
View tab. 43
Javadoc tab 44
Requirements tab 44
Beantab 44
Object Gallery 45

Setting Your Personal Preferences 49

Setting up workspaces 49
Accessing different workspaces 49
Managing workspaces 49
Saving workspaces 50
Using view management. 50
Controlling the level of detail shown in diagrams 51
Controlling how members display in diagrams . 51
Showing and hiding subpackage contents in diagrams 51
Controlling how JavaBean properties display in class diagrams 51
Showing and hiding referenced classes in diagrams 52
Showing dependencies between classes and interfaces 52
Controlling the display of sequence diagrams 52
Banning destinations in class diagrams. . . . 52
Showing or hiding aggregations of diagram and EJB elements 53
Setting JBuilder Integration options 53

Configuring Options 55

Activating and deactivating modules 55
Overview of configuration levels 56
Setting options in default mode 56
Setting options in advanced mode 58
Viewing and editing options 60
Values indicated by checkboxes 61
Accessing context-sensitive help for nodes and options 61
Resizing the Options dialog 61
Encoding of the configuration options . . 61
Reference guide to options. 61

Default and project level options 62
Diagram level options 66
Common Configuration Tasks 66
How to make Association links display a directional arrow 66
How to change the default source file header for the generated code 67
How to configure Stereotypes 67
How to customize Inspector properties . . 67
How to hide and show elements 67
How to enable mouse-wheel support (Windows only) 68
Configuring Together for improved performance . 69

Creating and Managing Projects 71

Working with Projects 73
Understanding Project Basics 73
Primary root directory 74
Project file 74
The <default> diagram 74
Creating new projects 75
Using the New Project dialog 75
Editing project properties 76
JBuilder-specific options 78
Synchronize method 78
JBuilder *project location* *79*
Adding Resources 79
Removing Resources 81
Setting Resource Options 81
Creating a new JDK configuration 82
Running multiple project instances 82
Setting up large projects 83
Creating views with referenced content 84
Performance tuning 84

Reverse Engineering 85
Reverse engineering techniques 85
Creating a new project with existing code . . . 85
Recognizing links during reverse engineering . . 87
Reverse engineering archive files 88

Importing and Exporting Information 91
Overview of import and export operations 91
Importing and exporting JDBC database information 92
Activating database import and export 92
Generating DDL (exporting to DDL) 93
Importing database meta-information 94
IDL Export and LiveSource 95
Exporting IDL from class diagrams 95
LiveSource support for IDL 96
Importing and exporting XMI models 96
Importing XMI models 96
Exporting XMI models 97
User configurable translations 97
Exporting a Together project to a JBuilder project 98

Modeling with Together 101

Introduction to Modeling 103
Overview of modeling 103
Diagrams supported in Together 104

Working with Diagrams 107
Creating diagrams in projects 107
Using the main menu or toolbar to create diagrams 108
Using the hyperlink feature to create diagrams 108
Cloning diagrams 109
Renaming diagrams 109
Configuring diagram options 109
Options for improving graphics 110
3D look 110
Anti-aliasing 110
Drawing diagram elements 110
Undo/Redo 111
Using the grid 111
Drawing elements 111
Configuring element symbol size 112
Adding multiple elements 113
Drawing relationship links 114
Drawing links with bending points 115
Drawing a link to self 115
Drawing dependency links 115
Filtering out autodependency links 116
Placing shortcuts on diagrams 116
Working with elements and links 117
Selecting elements 117
Right-click menus 117
Drag-drop moving and copying elements . . 118
Full drag-and-drop support 118
Copying and cloning elements 118
Resizing node elements 119
Converting bidirectional links 119

Labeling links 120
Changing links 121
Viewing diagrams and managing diagram layout . 121
Viewing 122
Zooming 122
Using the automated layout features 122
Creating your own manual layout 123
Tips and tricks 123
Editing diagrams 123
Opening and closing diagrams 123
Editing properties 124
Opening the Inspector 125
Modifying properties 125
Applying changes 125
In-place editing 126
Hyperlinking diagrams 127
Why use hyperlinking? 127
How to create hyperlinks 128
Hyperlinking to a new diagram 128
Hyperlinking to an existing diagram or diagram element 128
Hyperlinking to a URL or file 129
Viewing hyperlinks 129
Browsing hyperlinked resources 129
Removing hyperlinks 130
Annotating diagrams 130
Using Notes 130
Note Links 131
Inspector documentation tabs 131
Standalone design elements 131
Creating standalone design elements 131
Using standalone design elements 132
Saving and copying diagram images 132
Copy - paste within Together 132
Copy image 133
Save image 133
Printing diagrams 133
How to print diagrams 134
Using auto-layout for printing 134
Printing generated documentation 134
Troubleshooting 134
Tips and tricks 135

UML Modeling 137

Notation and stereotypes of UML diagrams . . 137
Use case diagrams 138
Diagram elements 138
Creating browse-through sequences of use case diagrams 139
Adding extension points 140
Class diagrams 140
Diagram elements 141
Physical (package) and logical class diagrams . . 142
Working with logical class diagrams 142
Working with package diagrams 143
Opening packages 143
Deleting packages 143
Renaming packages 143
Viewing packages and their content 143
Moving and copying elements from packages 143
Showing classes on search paths 144
Showing JavaBeans 144
Defining inner classes 144
Creating associations, aggregations, and compositions 145
Editing members and properties 146
Adding and editing members 146
Adding and editing properties 146
Rearranging the order of attributes or operations 147
Setting compartment controls 147
Sequence and collaboration diagrams 147
Diagram elements 149
Converting between sequence and collaboration diagrams 150
Creating sequence and collaboration diagrams . 150
Generating sequence diagrams from operations 150
Setting options to control generation of sequence diagrams 151
Specifying classes for objects 151
Working with messages 152
Messages in collaboration diagrams 152
Messages in sequence diagrams 152
Creating a message-to-self 152
Creating a message link that corresponds to an operation call 152
Changing activation times 153
Reordering message links 153
Branching messages 153

Annotating message links 154
Nesting messages 154
Creating statement blocks 155
Working with lifelines 155
Adjusting the default lifeline of objects 155
Adjusting the size of object lifelines. . 155
Changing the order of objects in a sequence diagram 155
Marking a destroyed object with an "X" . 155
Generating implementation source code . . 156
Statechart diagrams 156
Diagram elements 157
Drawing a self-transition 157
Creating internal transitions 157
Specifying entry/exit actions for a state . . . 158
Creating nested substates 158
Showing multiple transition sources or targets . 158
Modeling complex states 159
Activity diagrams 159
Diagram elements 159
Working with activity diagrams 161
Component diagrams 162
Working with component diagrams 163
Deployment diagrams. 164
Diagram elements 164
Working with deployment diagrams 165

UML Extension Diagrams 167

Business process diagrams 167
Diagram elements 168
Notation 168
Robustness analysis diagrams 168
Diagram elements 169
Key elements and properties 170

Generating Documentation 171

Generating Documentation for Together Projects 173

Generating documentation for an open project 173
Generating HTML documentation 174
Generate HTML documentation generation parameters. 174
Printing documentation 175
Print Documentation generation parameters 175
Generating documentation using a template. 176
Documentation generation parameters . . 177
RTF documentation options 178
Using a custom formatting template (.dot file) with DocGen. 179
Generating reports with the Rational® RequisitePro® plug-in 180
Setting up the reporting tool 180
Running reports 181
Documentation Generation Tips. 181
This section provides tips that may be used when generating documentation for your projects. 181
Adding package-level documentation . . 181
Line breaks in PDF documentation 182
Automating documentation generation 182
Launching the documentation generator from the command line 183
Documentation generation options 184
Options for generating documentation using a template. 184
Options for generating HTML documentation 185

Designing Custom Documentation Templates 187

Template Organization. 187
Body sections and the DocGen engine 188
Body section types 189
Metamodel types 190
Creating new templates 192
Setting template properties. 193
Designing the template body 194
Creating, arranging, and resizing sections . . 194
Setting section properties. 195
Output Style 195
Other options (Settings tab for folder sections) 195
Enabling Conditions 195
Accessing model elements through iterator sections 196
Element iterators 196
Element iterator scopes 197
Sorting 198
Element property iterators 198
Folder sections 199
Reusing templates and stock sections 200
Calls to stock sections 200

Creating calls to stock sections 200
Creating and editing stock sections 200
Calls to template sections 201

Documentation Template Controls 203
Using static sections, headers, and footers . . . 203
Creating and deleting static sections, headers, and footers 204
Setting area properties 205
Creating controls and setting control properties 205
Labels, images, and panels 205
Labels 205
Images 206
Panels 207
Data controls 207
Data Control Sources 207
Displaying custom properties 208
Formula and text controls 209
Formula controls 209
Include text controls 210
Managing the display of output 210
Moving and resizing controls 210
Text format settings 211
Aligning controls 212
Hyperlinking controls to element documentation . 212
Making a target from a static section, footer, or header 212
Linking a control to a target 213

Creating Multi-Frame HTML Documentation 215
Multi-frame HTML document template basics . 215
Frameset document template organization . 216
Creating frameset templates 216
Defining the frameset structure 216
Specifying frameset properties 217
Specifying frame properties 218
Designing the frameset template body 218
Frameset template body organization 218
Frameset template processing 219
Call to template section properties 219
Naming the generated document 219
Naming the output directory 220
Creating hypertext links (advanced) 221
Assigning a target frame to a link reference . 221
Targeting an element's "specific" documentation 221
Image mapping diagram elements 222
Creating compound link references 223
Javadoc link references 223
Converting JDRefs into hyperlinks 223
Converting {@link} tags 224
Converting the value of an element's property 224

Documentation Generator and Template Designer Reference 227
Documentation generator variables 227
Documentation generator functions 230

Contemporary Software Practices 243

Refactoring 245
Enabling Refactoring 245
Showing code structure 246
Moving classes, interfaces, attributes, and operations 247
Moving classes and interfaces 248
Moving attributes and operations in the class hierarchy 248
Renaming 250
Encapsulating attributes 251
Extracting interfaces and superclasses 252
Extracting interfaces and superclasses 252
Summary of refactoring commands 253

Templates and Patterns 255
Code templates 255
Template properties 256
Using template macros 257
Default properties in templates 257
Browsing available code templates 257
Editing code templates 259
Applying source formatting 259
Custom code templates 259
Working with the Code Template Expert . . .260
Creating custom code templates manually . .261
Groups of templates 262
Displaying custom template names 262
User-defined macros 263
Creating templates with custom file extensions . 264
Defining the source for extension of the newly generated files 264
Defining the custom template extension .264
Creating files with custom template extensions 264

Patterns 265
What are patterns? 265
Elements generated by patterns 266
Pattern behavior 266
Creating patterns 266
Providing pattern descriptions 268
Using templates and patterns 268
Creating classes and links by patterns 269
Creating a stub implementation. 270
Creating members by pattern 270
Choosing a pattern for a Member 271
Refactoring with patterns 272

Audits and Metrics 273

Overview of audits and metrics analysis 273
How should metrics be used?. 274
Running audits and metrics in Together 275
How to perform metrics analysis 276
Metrics for audit violations 276
How to perform audits 276
Options for processing audits and metrics . 277
Output for audits and metrics 277
Sorting results 277
Updating results 278
Exporting results 278
Creating linked HTML reports for metrics . 279
Saving and loading metrics results 280
Comparing metrics results. 280
Graphic output of metrics results 282
Kiviat graph 282
Bar graph. 283
Distribution graph 284
Printing results 285
Printing results from the table 285
Printing graphic results. 286
Automatic correction of audit violations . . 286
Creating and using saved sets of metrics or audits 287
Audits and metrics supported for Java 288
Running audits and metrics from the command line 292
Usage. 292
Options. 293
Extending the QA module 294
Additional information sources 295

J2EE Development 297

J2EE Support 299

Supported J2EE technologies 299
Special J2EE diagrams 300
Creating a J2EE diagram 301
EJB assembler diagram 302
Web application diagram. 302
Enterprise application diagram 302
Taglib diagram. 303
Application client diagram 303
Resource adapter diagram 303
References support 304
Security support. 304

Creating EJB Components 307

Setting the EJB properties for a project 307
Creating EJB components 308
Setting component editing and display options . 308
Setting component synchronization options .309
Creating EJB components from the class diagram 312
Creating EJB components using the Object Gallery 312
Applying an EJB pattern to an existing class .313
Importing entity EJBs from a database 313
Reverse engineering EJB source code 315
Using EJB Inspectors to set EJB properties 315
Configuring EJB inspectors. 315
EJB Inspector tabs 315
General tab 316
References tab 317
Deployment properties tab 318
Fields tab 320
Methods tab 321
Relationships tab. 322
Setting additional deployment properties 322
SunEE AS Properties Inspector tab 322
Borland Enterprise Server 5.2 Inspector tab .323
Borland Enterprise Server 5.2 Inspector tab for session EJBs. 323
Borland Enterprise Server 5.2 Inspector tab for entity EJBs 323
Borland Enterprise Server 5.2 Inspector tab for message-driven EJBs 324
WebLogic Inspector tabs 324
WebLogic 6.0 and WebLogic 6.1 Inspector

tabs 324
WebLogic 7.0 Inspector tabs 324
WebLogic 7.0 Inspector tab for session EJBs 324
WebLogic 7.0 Inspector tabs for entity EJBs 325
WebLogic 7.0 Inspector tabs for CMP entity EJBs 325
WebLogic 7.0 Inspector tabs for message-driven EJBs. 327
Working with EJBs. 327
Entity beans 327
Default entity bean code 327
Setting the persistence 328
Creating primary keys 329
Creating container-managed relationships . 329
Session beans 330
Default session bean code 330
Making the session EJB stateful or stateless . 331
Message-driven beans 332
Setting the project properties for message-driven beans 332
Default message-driven bean code 332
Setting deployment properties for message-driven beans. 333
Sharing home and remote interfaces. 333
Creating a second implementation class . 333
Deleting implementation classes with shared interfaces. 335
Customizing the default EJB code 335

Assembling EJB Modules 337

Visual assembly of EJB applications 337
Creating and modifying EJB modules 338
Creating an EJB module 338
Placing elements on an EJB assembler diagram. 339
Setting properties of an EJB assembler diagram 339
Displaying EJB shortcuts on EJB assembler diagrams 340
Creating EJB shortcuts 340
Displaying EJBs on EJB assembler diagrams 341
Showing and hiding EJB component parts . 342
How to copy and paste EJB shortcuts 342
EJB verification and correction 343
Verification overview 343
Using verification and correction 343
Deploying an EJB module 344

Container Transactions and EJB References 347

Container transaction elements 347
Container transaction attribute types 347
Creating container transaction elements . . .348
EJB references 350
EJB reference diagram elements 350
EJB properties elements. 350
Reference attributes in the client. 351
Supported EJB reference types 352
Reference attribute inspectors 352
EJB reference diagram elements 353
EJB resource and resource environment reference diagram elements 355
Creating EJB environment references 356

Designing and Developing Web Applications 359

Working with Web application diagrams 359
Creating a Web application diagram 360
Web application diagram elements 361
Servlet element 362
Web application diagram properties 363
General tab 363
Login Config tab 364
App event listeners tab 365
Creating shortcuts to applets, EJBs, EJB properties, container transactions, and references. .365
Developing applets 366
Creating an applet. 366
Deploying an applet 366
Working with servlets 366
Creating a servlet 367
Putting a servlet into a Web application . . .369
Working with JSPs 369
Working with tag libraries 370
Creating tag handlers on class diagrams . . .370
TagLib diagrams. 371
Diagram elements 371
Taglib diagram properties 371
Properties of the handlers. 372
Creating and using tag libraries 372
Working with filters 373
Creating filters and filter mappings 373

Setting filter and filter mapping properties . 374
Deploying filters 375
Working with listeners 375
Supported event types 375
Creating a listener 376
Associating listeners with Web applications 376
Deploying a Web application. 377

Designing and Developing Enterprise Applications 379

Visual assembly of enterprise applications for deployment 379
Creating enterprise application diagrams . 380
Enterprise application diagram elements . . 381
Properties of an Enterprise Application diagram 381
Creating diagram shortcuts 381
Importing existing archive files. 383
Deploying an enterprise application. 384

Designing and Developing Application Clients 387

Modeling enterprise application clients 387
Creating application client diagrams 388
Visual design elements 390
Properties of an application client diagram . 390
Creating "one-click" application clients . . . 390
Using an application client diagram 391

Designing and Developing Resource Adapters 393

J2EE Connector Architecture 393
Creating a resource adapter diagram 394
Resource adapter diagram elements 394
Creating a resource adapter diagram using a pattern 395
Using a resource adapter diagram 397

J2EE Deployment 399

Deployment process 399
Supported application servers 400
Integrating other application servers. 400
Generic server options 401
Installations and resources required for deployment 401
Starting application servers from Together . . . 402
J2EE Deployment Expert features 404
Diagrams for deployment 404
Specifying parameters 404
Running the J2EE Deployment Expert 405
Main scenarios for using the J2EE Deployment Expert. 406
Configuration files. 407
Server-specific deployment information . . .407
Sun EE reference implementation application server 407
Borland Enterprise Server 5.2 408
IBM WebSphere Application Server. . . .408
BEA WebLogic Application Server 408
J2EE Deployment Expert options common to all servers 409
First page: process options 409
Sun EE and Borland Enterprise Server 5.2 process options. 412
IBM WebSphere process options. 412
BEA WebLogic process options 412
Sun EE common properties. 414
Borland Enterprise Server 5.2 common properties. 414
IBM WebSphere common properties . . .414
BEA WebLogic common properties 414
Verify / Correct Sources page 414
Simple JSP Client Generation page 415
Options for WebSphere 3.5 416
Options for WebLogic 5.1. 416
Options for Sun EE, Borland Enterprise Server 5.2, WebSphere 4.0, WebLogic 6.0, 6.1, 7.0 . 417
Command Line File with Instructions for Deployment page. 417
Server-specific J2EE Deployment Expert pages .418
Sun EE Deployment Properties page 418
Borland Enterprise Server 5.2 Run-time Deploy properties page 418
BEA WebLogic Run-time Deploy properties page. 419
IBM WebSphere 3.5 pages 421
The IBM WebSphere 3.5 deployment options differ significantly from those for other servers. 421
EJB Deployment Properties page 421
Servlet Deployment Properties page . . .422
Client Properties page. 424
IBM WebSphere 4.0 pages 425
Advanced Single Edition Properties page 425
Deployment Properties page 425
Making transitions among application servers .426

Transitions among specifications 426
Automatic verification and correction of EJBs 427
Manual correction of EJBs 427
Moving from EJB 2.0 to EJB 1.0 427
Moving from EJB 1.0 to EJB 1.1 428
Transitions among application servers . . . 428
Developing and deploying clients for various servers 429
Encoding in deployment descriptors 430

J2EE Platform Security Support 431
Security support in J2EE diagrams. 431
Security elements in J2EE diagrams 432
Security roles 432
Linking security elements 433
Security in EJB modules. 433
Declarative security and method permissions . 434
Programmatic security and security role references 435
Security in Web applications 437
Security roles and principals 437
Constraints and Web resource collections . . 438
Security in enterprise applications. 440
Security in resource adapters. 440

Extending Together 443

Advanced Customizations 445
Customizing the user interface 445
Creating custom diagram types 446
Step 1: Creating icons and icon references for the UI. 447
Editing the Together.bat file 447
Referencing images in the config file . 447
Step 2: Creating the diagram configuration file. 447
Step 3: Defining tree-view icons 448
Step 4: Defining toolbar icons. 448
Step 5: Defining viewmaps 450
Defining graphical presentation 450
Defining the shape of an element 451
Defining the element's options 451
Enable display of the element's unique name 451
Add the element's name to the compartment 451
Enable in-place editing 452
Define label properties. 452
Define links 452
Sample configuration file. 452
Customizing Property Inspectors 453
Adding custom pages and fields to the Inspector 454
API-based Inspector customization 454
More API documentation 456
Customizing Inspector by means of the Custom Properties Module 457
User-defined inspector customization with the Inspector Property Builder 457
Customizing View Management *Show* options .458
Changing the display text of a Show option .458
Removing a Show option from the Options dialog. 459
Adding a Show option to the Options dialog 459

Together Open API 461
Overview of the Together Open API 461
Understanding modules 462
Locating modules 463
Registering a module 464
Starting modules 464
Examples for Hello World 464
Extending Together using modules. 465
Types of modules 466
Interfaces implemented by the modules . . .466
Viewing modules 466
Running modules 467
Guidelines for developing modules 467
Applying naming conventions. 468
Documenting the module 469
Order of Tags 469
Deploying the module 470
Tutorial: Writing and deploying modules 470
Source code for the module 470
Declaring a module 472
Declaring a module in a Manifest.mf file .472
Declaring a module in a .def file 475
Specifying whether a module is a startup module 475
Defining a visible name for the module . . 475
Specifying the location of the Modules tab 475

Adding the module name to menus for classes, interfaces, and members . . . 476
Adding the module name to menus of elements with specific shapetype 476
Rules for the manifest file 477
Compiling the module 477
Storing the compiled class 477
Evaluating the results 478
Troubleshooting your module 478
Accessing more API documentation 480

Configuring Together for Multiple Users 481

Overview of a shared multi-user configuration 481
Setting-up multiple workstation installations . 482
Creating the start-up pointer file 483
Launching via the command line and pointer file 484
Adding new configuration levels to predefined levels 484
Adding new configuration directories to a shared location 485
Adding new properties directories to a local installation 485
Creating the path.config file 485
Viewing added configuration levels 486
Modifying default configuration levels 486

Commands, Macros, and Keyboard Shortcuts 489

Command Line Parameters 489
Basic command line syntax 489
Using the Windows launcher 491
Invoking the Together main class 491
Command-line examples 492
Documentation generation 492
Running modules at start-up 493
Parameters for the Together.exe Launcher 493
Parameters for virtual machine preferences .493
System Macros 494
Referencing configuration properties as macros 497
Template Macros 498
Keyboard shortcuts 499
Main Window and Main Menu 499
Designer shortcuts 501
Miscellaneous 502

Index 505

PREFACE

The *User Guide* contains instructional and conceptual information for using Together® Edition for JBuilder® to build enterprise level applications. The guide assumes that you have installed Together on your computer. If not, refer to the Readme.

Audience

The information in this guide is intended primarily for developers using Together to build applications. The documentation assumes that you have experience with the following:

- The basics of the Unified Modeling Language (UML).
- Your organization's process for designing and modeling object-oriented systems or components.
- A working knowledge of Java.
- Fundamentals of development technologies such as Enterprise JavaBeans (if you will use Together to develop and deploy an EJB, JSP, etc.).
- How to install a Java VM and launch a Java application with it.

Together Documentation Set

The documentation set for Together consists of the items listed in Table 1.

Table 1 Together documentation set

Item	Description	Location
Together Edition for JBuilder Readme	Late-breaking information including: • Licensing notes • System requirements • Installing and starting Together • Known problems • Significant fixes	Your Together installation under `$TGH$/readme.html`

Table 1 Together documentation set (continued)

Item	Description	Location
Getting Started with Together Edition for JBuilder	Information for using Together Edition for JBuilder for the first time, including: • Setting up your directory structure for Together and JBuilder • Creating a project in Together Edition for JBuilder • Other information specific to using Together Edition for JBuilder	PDF located as follows: • Together's main menu under **Help** | **Manuals** • Your Together installation under `$TGH$/doc/gettingStarted.pdf`
User Guide	Comprehensive information most relevant to the user, including: • Introduction to Together • Setting personal preferences and options • Detailed instructions for using Together features • Advanced customizations • Reference information including commands, macros, and keyboard shortcuts	PDF located as follows: • Together's main menu under **Help** | **Manuals** • Your Together installation under `$TGH$/doc/userGuide.pdf` • http://info.borland.com/techpubs/together/
Online Help	A subset of the information in the *User Guide*, including: • Main window components • Brief instructions for using Together features with references to the *User Guide* for more details • Dialog help	Together's main menu under **Help** | **Application Help Contents**

Finding out more

If you are just getting started with UML, object technology, or distributed application development, the following books and resources can help get you started.

Third-Party Books

UML Distilled: Applying the Standard Object Modeling Language by Martin Fowler. Addison-Wesley, 1997. ISBN: 0201325632

The Unified Modeling Language User Guide by Booch, Rumbaugh, and Jacobson. Addison-Wesley, 1998. ISBN 0-201-57168-4

Java Modeling in Color with UML: Enterprise Components and Process by Peter Coad, Jeff De Luca, and Eric Lefebvre. Prentice Hall, June 1999. ISBN: 013011510X

Java Design: Building Better Apps & Applets (2nd Ed.) by Peter Coad and Mark Mayfield. Prentice Hall, 1997. ISBN 0-13-271149-4 (pbk).

Java Enterprise in a Nutshell by David Flanagan, Jim Farley, William Crawford, and Kris Magnusson. O'Reilly & Associates, 1999. ISBN 1-56592-483-5

On the Web

Object Management Group: http://uml.shl.com/

Cetus Links site: http://www.cetus-links.org

Dev-x Developer Exchange: http://www.devx.com/

Part

1

GETTING STARTED

- Chapter 1, “Tour of Together”
- Chapter 2, “Setting Your Personal Preferences”
- Chapter 3, “Configuring Options”

Chapter

1

Tour of Together

Together Edition for JBuilder includes the features you need to design enterprise level applications, such as UML modeling, patterns, QA audits and metrics, and documentation generation. This version of Together is intended for use with JBuilder. The LiveSource™ feature keeps your Together diagrams synchronized with your source code in the JBuilder editor.

This chapter provides an introduction to the basic features of Together. It includes the following topics:

- "Getting started with a sample project" on page 21.
- "Navigating the Together main window" on page 25
- "Explorer pane" on page 30
- "Designer pane" on page 38.
- "Message pane" on page 39.
- "Property Inspectors" on page 40.
- "Object Gallery" on page 45

Getting started with a sample project

Almost all work in Together is done in the context of projects. Together ships with a collection of sample Java projects in your Together installation under `$TGH$/samples`.

The sample Java *CashSales* project is the basis for the tutorial steps in this section.

Opening the CashSales project in JBuilder

Both a JBuilder and Together project must be created to successfully use the two products together. To use the *CashSales* project for the tutorial steps in this section, we must first create this project in JBuilder.

To create the CashSales project in JBuilder:

1. From JBuilder's main menu, choose **File | New Project**. The Project Wizard displays.
2. Enter `CashSales` for the *Project Name*, and click **Next**.
3. Under the *Source* tab, click **Add**. Add the following source path: `$TGH$\samples\java\CashSales`, and click the *Default* radio button.
4. Click **Ok** to create `CashSales.jpx`.

Opening a project

You can easily open Together while working in JBuilder. There are many ways to create and open projects, but using the Project pane right-click menu is the quickest method.

To open the CashSales project in Together:

1. In JBuilder, right click on the project root in the Project pane, and choose **Open project in** Together.
2. If Together is not running, a dialog displays allowing you to start the application. Click **Yes** to continue. Note that Together allows one project to be open at a time. If another project is currently open in Together, a dialog displays, advising you that Together is running a different project.
3. Switch focus to the Together application to continue.

You can also open both Together and JBuilder projects from within Together.

To open any type of project in Together:

1. In Together, choose **File | Open Project**. The Open Project dialog opens.
2. In the *Files of type* drop-down list at the bottom of the dialog, choose *All projects*. This filter displays all recognized project file types in the navigation tree, including Together (*.tpr) and JBuilder (*.jpx) project files.
3. Navigate to the project file you want to open and click **Ok**.

Notice that once you have opened a project, the project location and name is added to the list under **File | Recent Projects**. You can choose a project from the list to open it. Use this method as a shortcut for opening projects that you have already worked with. Recent Projects saves a list of all the projects you open during a Together session. The next time you start Together, the Recent Projects list contains the last 10 projects you opened.

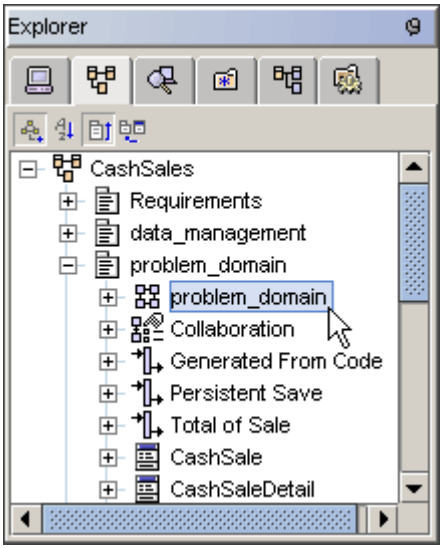
Opening a diagram

By default, the Explorer pane occupies the upper left side of the main window. You can open diagrams and source code files from the Explorer pane.

To open the `problem_domain` *diagram:*

1. Click the Model tab ( ) at the top of the Explorer pane.
2. Expand the model tree to find the diagram: **CashSales** | **problem_domain**.
3. Right click the **problem_domain** diagram. (See Figure 1.)
4. Select **Open in New Tab** from the menu. The diagram opens in the Designer pane.

Figure 1 Selecting the **problem_domain** diagram from the Explorer.



Observing correlation between model and code

You can open source code files in the JBuilder Editor from both the Explorer and Designer panes.

To open CashSale.java from the Designer pane:

1. Find the *CashSale* class node in the *problem_domain* diagram in the Designer.
2. Right click on **CashSale**, and choose **Edit**. Switch focus to JBuilder. `CashSale.java` displays in the JBuilder Editor.

When you double-click on a member of a class node in the ExplorerDesigner, upon switching focus to JBuilder the declaration of that member highlights in the JBuilder Editor.

Together always keeps model and code in sync. If you change the model, the code changes. If you change the code, the model changes.

To observe how changing a diagram also changes code:

1. Double click the attribute named *SALES_NEW* in the *CashSale* node of the *problem_domain* diagram.
2. Enter a new name and new type for the highlighted attribute.
3. Right click the member, and choose **Edit**. Switch focus to JBuilder. Notice how the declaration of the attribute changes in the JBuilder Editor.
4. Return focus to Together. Undo the change by clicking the undo button ( ) on the main toolbar or pressing Ctrl+Z.

To observe how changing code also changes diagrams:

1. Using the JBuilder Editor, find the declaration of *SALES_NEW* in the file `CashSale.java`.
2. Rename *SALES_NEW* to *XXX*.
3. Return focus to Together. (The source code automatically updates; there is no need to save it.)
4. Observe the new name *XXX* in the *CashSale* node in the diagram.

Activating special features

Together has an extensive collection of special activatable features such as audits and metrics. Some of the features are activated by default. You must activate others that you want to use.

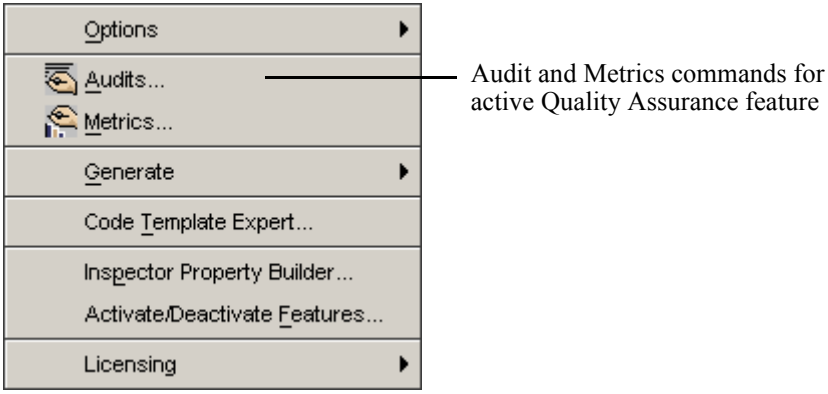
To activate the Quality Assurance feature for generating audits and metrics:

1. Select **Tools | Activate/Deactivate Features** from the main menu.
2. In the resulting dialog box, bring the Together **Features** tabbed page to the front.
3. Click the **Quality Assurance** checkbox on.
4. Click **OK** to close the dialog box and turn on activation.

Notice that the Message pane displays an informational message when a feature module has been activated or deactivated. For information on opening and using the Message pane, see "Message pane" on page 39.

Figure 2 shows the Tools menu with its new *Audits* and *Metrics* commands.

Figure 2 Tools menu with Quality Assurance activated



Navigating the Together main window

The main window is divided into three major panes.

- **Explorer:** for file system and project navigation.
- **Designer:** for creating UML and other kinds of model diagrams.
- **Message pane:** for system messages, special tasks, and results of some feature operations.

The *focus pane*, with the light blue title bar, is the site of the most recent activity.

Together *elements* are individual components of the project and the user interface. Elements can be diagrams, files, diagram elements such as nodes or links, names, error messages, and so on. Right clicking on a Together element displays a menu of commands for that element. These right-click menus vary according to the type of element. Figure 3 shows the right-click menu for a note on a diagram in the Designer pane, which is undocked from the main window.

Many elements have *Inspectors* for accessing the elements' properties. You can display the Inspector of an element by selecting Properties from its right-click menu (see Figure 3).

Arranging panes and the Inspector

You can rearrange the windowing and layout of the panes and the Inspector. You can hide panes, resize them, and expand them to full screen. You can also undock panes from the main window or move them from one side of the main window to another.

View or hide a pane

You can view or hide any available pane and the Inspector.

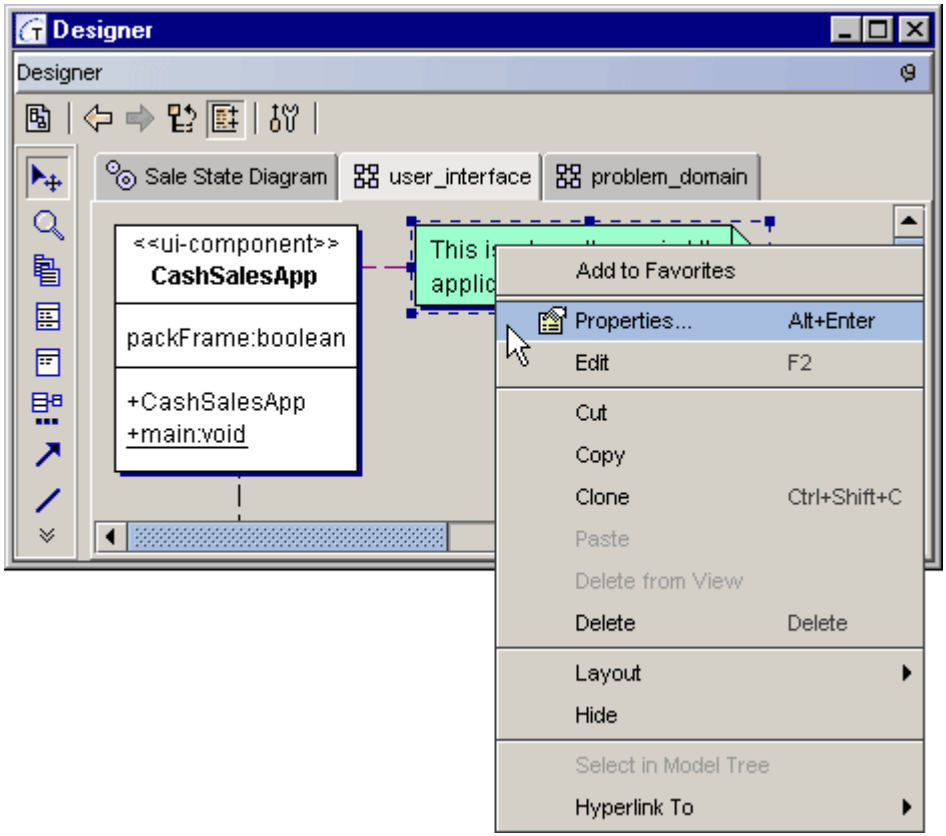
To view or hide a pane or Inspector:

- Click the corresponding view/hide button on the main toolbar (see "Main toolbar" on page 29), *or*
- On the main menu, choose **View | Main Panes | <name>**, *or*
- Type the keyboard shortcut for the pane. The keyboard shortcuts for the different panes are listed on the **View | Main Pane** cascading menu.

Buttons and names of panes that are not available are shown in gray on the main toolbar and menu.

The Designer pane is available only when there is an open project. The inspector is available only if the currently selected element has an inspector. (Tabs in the Explorer do not have inspectors. However, diagrams in the Designer pane do have inspectors.)

Figure 3 Accessing an inspector from a right-click menu



Resizing or expanding panes to full screen

To resize panes, drag their separating bars.

To expand a docked pane to full screen:

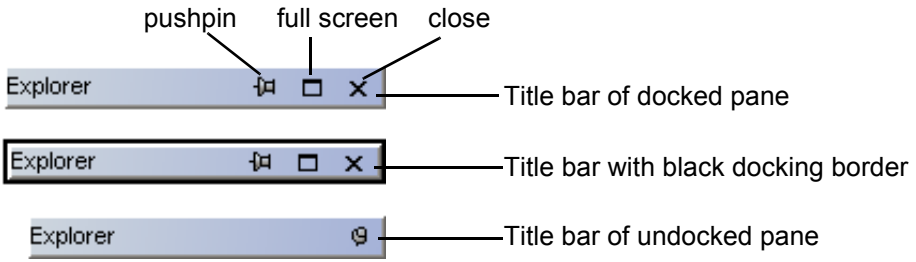
- Click its full screen expander box on the title bar (see Figure 4), *or*
- Select **Full Screen** from the right-click menu of its title bar, *or*
- Double-click the title bar, *or*
- Bring the pane into focus by clicking it *and*
  - click the **Full Screen** button on the main toolbar, *or*
  - select **View** | **Full Screen** from the main menu, *or*
  - press **F12**.

Repeat any of the actions listed above to return the pane to its previous unexpanded state.

Undocking and moving panes

Each pane in the main window has a push pin in the upper right corner of its title bar (see Figure 4). Clicking the push pin undocks the pane from the main window. Dragging the title bar off the pane also undocks it.

Figure 4 Title bar variations



To redock the pane back on the main window, click the push pin on the title bar of the undocked window. Alternatively, drag the title bar off the pane and onto the main window, releasing it when the border turns black. (See Figure 4.)

You can move the Explorer, Inspector, or Message panes from their original docked positions to either side of the main window. Simply drag the title bar as described above, releasing it when the border turns black.

Hiding pane title bars

Even the title bars of panes and the Inspector have right-click menus. The items on the right-click menu include those on the title bar itself; they change when switching between docked and undocked.

The last item on the right-click menu of each title bar is **Hide Title**. When you select it, the title bar becomes a narrow line at the top of the pane.

The main menu, toolbar, and status bar

The main menu and toolbar are at the top of the main window. The status bar is at the bottom.

Main menu

The main menu consists of a collection of menus, shown in Table 2.

Table 2 Main menu items

Menu	Meaning
File	Commands for interacting with the operating system. Use the File menu for the usual file commands as well as special commands such as Import/ Export.
Edit	Editing commands plus “infinite” undo/redo for almost all operations.
Search	Search and replace commands. Use the Search menu to find and replace text strings in source files, diagrams, and across multiple files in a project. You can also create custom queries for searching.
View	View options for the main window. Use the View menu to hide or show any of the panes. You can also maximize the currently selected pane. Note the keyboard shortcuts displayed on the menu.
Project	Commands and properties for the currently open project. Use Project to generate documentation and set project properties.
Deploy	J2EE deployment expert. Available only when the E-Commerce feature is activated.
Selection	Commands relevant to the currently selected element. Selection replicates the right-click menu of the currently selected element. The menu content changes dynamically to reflect commands appropriate to the selection.
Tools	Options (default, project, diagram), feature activation and special feature commands, code template experts, and licensing. The Tools menu varies according to which features are activated.
Help	Online documentation for Together, API documentation, web links, and setup for your personal context help libraries.

The menus vary according to the workspace, the status of the open project and project options, the activated features, and the currently selected element.

Some of the items on the individual menus have cascading submenus. Many commands on the menus have keyboard shortcuts. For a complete list of keyboard shortcuts, see “Commands, Macros, and Keyboard Shortcuts” on page 489.

Main toolbar

Main menu commands that are frequently used have corresponding buttons on the main toolbar.

Table 3 Main toolbar buttons

Icon	Name	Meaning
	New object	Displays the Object Gallery.
	New project	Closes the current project and displays the new project dialog.
	Open/Save/ Print	Common file operations.
	Undo/Redo	Available for most operations in the Designer.
	Cut/Copy/ Paste	Common edit operations.
	Find	Find in current file. (Search/Find across multiple files is available on the main menu.)
	View Explorer	Toggle to view or hide Explorer pane.
	View Designer	Toggle to view or hide Designer pane.
	Message pane	Toggle to view or hide Message pane.
	Full screen	Toggle to maximize the main window focus pane.
	Inspector	View properties Inspector for current selection (toggle). It is available only when the selected element has an inspector.
	Help	Launch online help system.

The right-most item on the main toolbar is a menu for selecting the current workspace. For information on workspaces, see "Setting up workspaces" on page 49.

You can undock the main toolbar from the main window by dragging the "handle" at the left edge. Click the close button on the undocked toolbar to dock it.

Status bar

The status bar for the main window varies according to which panes are docked and in view. The full status bar shows the following items, left to right:

- **Messages (button):** Shows or hides the Message pane; indicates when there is a new message.
- **General info:** Displays information about elements in diagrams as you move the mouse over them.
- **Progress:** Shows the progress of internal processes that you invoke while working in Together.

Explorer pane

The Explorer pane enables navigation of your system, provides detailed views of your project, and gives easy access to system and extension modules and reusable components. On the Explorer pane you can:

- View the physical and logical structure of your project.
- Navigate within the project as well as within your physical directory system.
- View project resources such as source files, classes and members, diagrams, and modules. Open diagrams and files for editing.
- View and run Together system and feature modules or any custom ones you develop.

The Explorer pane is organized into multiple tabs, as shown in Table 4. Some tabs display only with an open project or when special features are activated. By selecting **View | Explorer Tabs** from the main menu, you can toggle the display of three tabs: Server Explorer, Diagrams Explorer, and Favorites.

Table 4 Explorer pane tabs

Icon	Name	Meaning
	Directory	Physical structure of the open project and the file system. Does not require an open project.
	Model	Logical view of the project's model elements. Requires an open project.
	Server	Servers relevant to your Together installation. Does not require an open project; toggle on or off with **View	Explorer Tabs	Server Explorer**.
	Favorites	Frequently accessed ("favorite") model elements and files. Requires an open project; toggle on or off with **View	Explorer Tabs	Favorites**.
	Diagram	Listing of logical project diagrams by type. Requires an open project; toggle on or off with **View	Explorer Tabs	Diagram Explorer**.
	Components	Reusable model components. Requires an open project in which "include components" is checked on.
	Modules	Custom building blocks. Does not require an open project.

Directory tab

The Directory tab presents views of the contents of your physical system and enables you to navigate both inside and outside of a project's physical structure. In the Directory tab, you can:

- Find files

- Cut, copy, paste, and delete files and folders (right click and choose **Cut, Copy, Paste,** or **Delete**).
- Create a new project around a directory (right click the directory and choose **New Project**).
- Add a directory to the project path (right click the directory and choose **Add Root to Project**).

The Directory tab shows an icon beside each file, including these:

Edit file icon. Double click source code and text files to open in the JBuilder Editor.

Together project icon. Double click to open the project in Together.

JBuilder project icon. Double click to open the project in Together.

Generic file icon. Together cannot open these files.

When you launch Together, the Directory tab displays your available drives/directories and the following additional directories from your Together installation:

- **Samples:** Contains example projects.
- **User Projects:** Empty directory where you can place your first experimental projects or real projects for quicker access from the Directory tab. In the latter case, you should create the Together project under the JBuilder default project directory.
- **Templates:** Contains numerous templates of classes, members, and links.

  When you open a project, the Directory tab displays an additional directory at the top:

- **Current Project:** Physical files that comprise your project as well as files relevant to your project. This shows only when there is an open project.

Server tab

The Server tab enables you to navigate through the application and database servers relevant to your project and to your Together installation.

Model tab

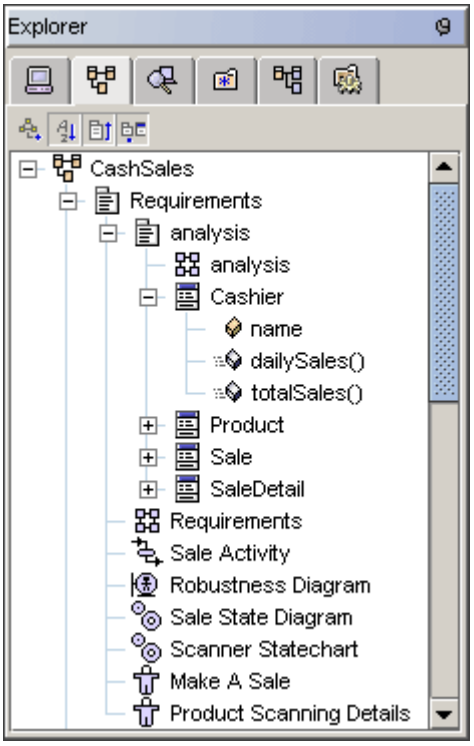
The Model tab becomes visible when you open a Together or JBuilder project. It displays a logical view of the elements that comprise the project's model. (See Figure 5.)

The Model tab displays the root-level packages that comprise your project, enabling you to logically navigate the contents to see the subpackages, diagrams, and diagram elements in each one.

Note The Model tab view is not strictly hierarchical in the same sense as a file system explorer because the project's root-level packages can physically reside anywhere on your system. The Model tab displays secondary root packages in relation to the primary root (the package containing the Together project file).

For example, you might create a project file in the directory `C:\Project1`, and specify `D:\mysources` as the secondary root directory for the project. In this case, `mysources` displays as a package node under the primary root node (`Project1`). From that point on, the contents of `mysources` displays hierarchically.

Figure 5 Explorer Model tab



Opening diagrams

From the Model tab, you can open diagrams in the Designer.

To open a diagram from the Model tab:

- Double click the diagram, or select **Open in New Tab** from the diagram's right-click menu to open the diagram without closing the diagram at the front, *or*

- Choose **Open** from the diagram's right-click menu to replace the diagram at the front if the Designer is already open. (You can set the project or default options to open in a new tab instead of replacing the front diagram.)

  - or -

- Choose **Select in Diagram** from the right-click menu of a diagram element (class, interface, operation, attribute) to open the containing diagram if necessary and to highlight the element on the diagram.

Note When you open a diagram from the Model tab by selecting a key element such as a class, member, or use case (depending on the diagram type) on the Model tab, the element is simultaneously selected in the diagram.

Tip *To open a package diagram from within JBuilder,* select the package in the Project pane, and choose **Open diagram in** Together.

Model, Favorites, and Diagram tab toolbar

The Model tab, Favorites tab, and Diagram tab display a small toolbar (see Figure 5). The toolbar buttons control the presentation of information in the tree-view. From left to right, these buttons are:

- **Expandable diagram nodes:** Controls whether you can expand nodes on the tab to show their contents.
- **Sort node:** Controls whether nodes in the tree-view are sorted alphabetically.
- **Packages first:** Controls whether packages display first in the tree-view before other content.
- **Show fully qualified names:** Controls whether fully qualified class and interface names are displayed in tooltips. If this toggle is off, short names are displayed.

Note *Show fully qualified names* is not intended for use with Java, because the names become too long as you move deeper into the package hierarchy. Therefore, in Java projects this option displays fully qualified names only for root packages with the package prefix set.

Favorites tab

The Favorites tab can display the nodes from the model that you want to access without navigating the logical project hierarchy in the Model tab or in a diagram. Nodes can be diagrams, classes, or interfaces. The Favorites tab is particularly useful for large projects. The rightmost button on the Favorites tab toolbar is a toggle for showing the fully qualified names of classes and interfaces.

You can add an element to the Favorites from the Designer as well as from the Model tab of the Explorer.

To add an element to Favorites:

1. On the Model tab or in the Designer, find and select the element to be added to Favorites.
2. Right click the element and choose **Add to Favorites**.

You open elements in Favorites in the same way you would open them from the Model tab.

To remove an element from Favorites:

1. Select the element in the Favorites tab.
2. Right click and choose **Remove from Favorites**.

Diagrams tab

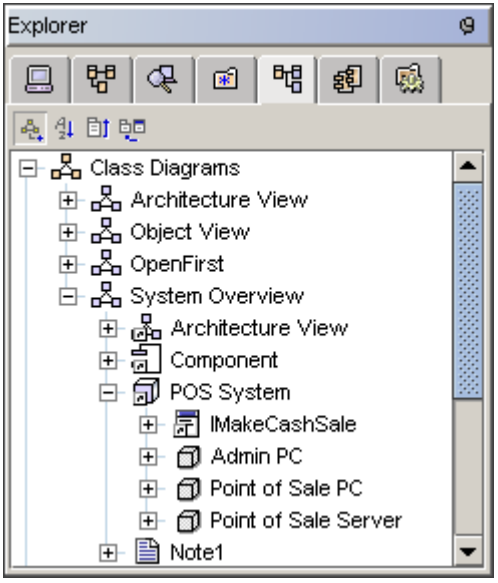
The Diagrams tab displays a list of all types of logical diagrams available in Together. The diagrams are listed according to their types. You can open the diagrams from the Favorites tab the same way as in the Model tab.

Note The Model tab does not display package diagrams. They are physical diagrams rather than logical diagrams.

When *Expandable diagram nodes* (on the Diagram toolbar) is on, you can expand each diagram to see its elements. Elements on diagrams that are not actually contained in the diagrams appear as *shortcuts.* For example, if a class diagram displays classes that reside outside the class diagram's package, the nodes for these classes have shortcut symbols in both the tree-view and the diagram. Class diagrams can show classes and interfaces from packages that reside on the classpath or other search paths as defined in the project's properties.

Figure 6 shows the Diagram tab and several shortcuts including POS System, and IMakeCashSale. Shortcut icons have the curled arrow in the lower left corner. Note that in this example the *Expandable diagram nodes* button on the toolbar is turned on, so diagram elements are visible. If this button is turned off, you will only see the list of diagrams in the project without any diagram content.

Figure 6 Diagram tab showing shortcuts



The Diagrams tab is available whenever there is an open project. You can leave it visible or hide it from view.

To hide the Diagrams tab:

1. Select **Tools | Options | Default (or Project) Level** from the main menu.
2. In the resulting Options dialog box, choose *General* without expanding it.
3. Clear *Show Diagrams tab*.
4. Click **OK** to close the dialog box and save the setting.

Modules tab

Together is highly extensible. Using Java and the Together API, you can develop your own modules to accomplish tasks such as creating custom metrics or documentation and generating custom outputs based on model information. Indeed, many of Together's own features are implemented as building-block modules. These modules appear in the system folder on the Modules tab.

The Modules tab provides quick access to system, sample, and any added-in building blocks that are supplied with Together or developed or added yourself. You can view the available building blocks by navigating through the Modules tab folders. You can run any compiled or source files from the right-click menu of individual nodes. If you develop your own modular building blocks or acquire third-party building-block extensions, you can install them so that they display in and run from this tab.

The Modules tab uses file icons as shown in Table 5.

Table 5 Module tab icons

Icon	Meaning
	Java source file for module. Can be compiled from "Run" when a compiler is configured.
	Compiled Java module.
	TCL script. "Run" executes the script in interpreted mode.

See Chapter 30, "Together Open API" for more information on working with modules.

Components tab

The Components tab enables you to access and reuse component models. It shows the `$TOGETHER_HOME$/modules/components` directory of your installation. This directory includes the "Coad Modeling Components," which are enterprise component models in color that you can reuse or modify. You can add your own components to the Components tab by placing them in directories under the components directory.

To display the Components tab for a newly created project:

1. Select **Tools | Options | Default (or Project) Leve**l from the main menu.
2. In the resulting Options dialog box, expand the *General* and *New Project* nodes.
3. Check *Include Components*.
4. Click **OK** to close the dialog box and save the setting.

You can copy classes and packages from the Components tab to your class diagrams, thus creating new source files and packages in your project. You can copy them to any package in your project as well. Classes and packages copied in this way appear on the Model tab in the appropriate package but do not appear in any class diagrams until you open the diagram and the source files are reverse engineered.

"Explorer pane operations" on page 36 describes how to copy between Explorer tabs and diagrams.

Explorer pane operations

There are several operations that are common across two or more Explorer pane tabs.

Right-click menus

The right-click menus on various types of nodes on Explorer tabs enable actions such as opening source files, and performing clipboard operations. Explorer nodes for classes, use cases, and other main diagram elements have the same right-click menus as the elements in the diagram. You can perform the same operation from either place with identical results.

Quick project access

The default directory for newly created projects matches that of JBuilder. For example, on a Windows system, your default project directory is:

```
../Documents and Settings/<user_name>/jbproject
```

When you create project directories, they display under the `User Projects` folder of the Directory tab. You can quickly navigate to projects and open them from there.

To customize the default location for new projects:

1. Select **Tools | Options | Default Level** from the main menu.
2. Expand the *General* node.
3. Choose the *New Project* node without expanding it.
4. Enter the ***Default location*** in the textfield on the right.
5. Click **OK** to close the dialog box and save the setting.

Copying and pasting

You can copy and paste classes, packages, diagrams, and elements of diagrams using the Explorer. Items that can be copied have a Copy command on their right-click menus. You can copy between the Component and Model tabs or from the Explorer to an open diagram.

To copy something from the Explorer:

1. Select the Explorer tab containing the source.
2. Select the items you want to copy.

Note To select several items at once, use Ctrl+click.

3. Choose **Copy** from the right-click menu.
4. Choose **Paste** from the right-click menu of the destination, which can be a package in the Explorer or a diagram in the Designer.

It is also possible to clone diagrams, elements, and class members. Cloning creates an exact copy of the cloned item in the same package or node element and gives it a default name, which you can edit.

The instructions for cloning are identical to those for copy, except select **Clone** from the right-click menu.

Note You can also clone node elements in the Designer pane using their right-click menus.

Finding the locations of nodes, files, and folders

On the Directory, Model, Modules, Diagrams, and Components tabs, you can navigate to the desired location without actually scrolling the tree-view. With the appropriate tab active, type the name of the desired folder. As you enter the characters, the highlight moves to the appropriate node of the tree-view.

There is no need to type in complete names of nodes. As soon as the required node is reached, type a delimiter (slash, backslash, or dot) to complete the name and expand the node; then continue typing the name of the required nested node. Keep in mind that the entry is case sensitive.

Designer pane

The Designer pane is visible only when there is an open project. Use this pane for creating UML and other kinds of diagrams that make up the project model.

Opening and closing diagrams

You can open a diagram from the current project from the Model tab of the Explorer pane (see "Model tab" on page 31).

To close a diagram:

- Right-click on the diagram background and select **Close**, *or*
- Press **Ctrl+F4**, *or*
- Right click the tab and select **Close** or **Close All**.

The Designer shows a tab for each open diagram. The tab displays an icon for the UML diagram type and the name of the diagram. The *focus diagram* is the one at the front. To switch between open diagrams, click the tab of the one you want to view.

Designer pane toolbar

The Designer pane toolbar lies across the top of the pane. The toolbar enables you to:

- navigate between different diagrams
- create new diagrams
- manage the appearance of the diagrams

Tip You can hide the toolbar on the Designer by right-clicking the title bar and selecting **Hide Toolbar**.

Table 6 Diagram toolbar icons

Icon	Name	Meaning
	New diagram.	Displays the Object Gallery dialog box. From there you can select the new diagram type, name, location, and description.
	Back	Returns to the diagram from which this was first reached.
	Forward	Goes to the diagram first reached from this one.
	Parent	Returns to parent diagram.
	Expandable compartments	Makes the compartments for class and interface nodes expandable.
	Diagram view management	Displays view management options for showing or hiding different types of diagram content.

Designer tools

The tools for constructing diagram elements lie on a toolbar on the left side of the Designer pane. You use the tools to draw elements on the Designer pane to represent the structural and behavioral elements and interactions of your model.

The Diagram Design tools vary according to the type of focus diagram. These tools include principal elements defined by the UML for the focus diagram type as well as Together-specific enhancements (such as *Class by Pattern* in the class diagram). Mouse-over tool tips identify the toolbar icons. *Note* and *Note Link* elements are common to all diagram types.

Message pane

The Message pane displays pages that:

- Provide system information. The Messages page shows a queue of system messages. Use the right-click menu to save or copy messages, navigate to problem spots described by a message line, or clear the message queue.
- Display the results of operations.

Messages are organized into tabbed pages, with a separate tab for each type of task, operation, or source of information.

You can show or hide the Message pane by using the main toolbar or the equivalent keyboard shortcut, Ctrl+Alt+M. When the pane is hidden, an icon in the first cell of the status bar indicates the presence of messages in the message queue, as shown in Table 7.

Table 7 Status bar icons for the Message pane

Icon	Meaning
	Message pane is open.
	Message pane is closed, no new messages.
	Message pane is closed, new messages in queue.

To remove a message from the queue in the Message pane:

1. If the Message pane is closed, open it.
2. Right click the message and select **Remove** (to remove a single message) or **Remove All** (to remove all messages).

The Message pane right-click menu also enables you to sort system messages (by time or type), filter them, and save them to a file.

To configure the maximum number of messages in the Message pane:

1. From the main menu, choose **Tools | Options | Default Level**.
2. In the resulting Options dialog box, choose the *General* node.
3. Enter the maximum number of messages to display in the *Messages Maximum Count* field.

Note You can also set the Message pane to open automatically when a new error is detected. Choose the option called *Open Message pane on error*.

4. Click **OK** to close the dialog and make the changes.

Property Inspectors

Property inspectors enable you to view and change the properties of many Together elements. Inspectors are organized into tabbed pages whose content depends on the type of element.

To display the properties of an element:

1. Select the element or diagram in the Designer pane or in the Explorer.
2. Do one of the following:
   - Right-click and choose **Properties** from the right-click menu, *or*
   - Use a keyboard shortcut (Alt+Enter or Alt+double-click), *or*
   - Click the **Properties** icon on the main toolbar, *or*

- Choose **Selection | Properties** on the main menu.

When the Inspector is already open, selecting an element automatically loads its properties into the Inspector.

When multiple elements are selected at the same time, the Inspector displays only those properties that are common to all the selected elements.

Overview of Inspectors

The properties that Inspectors enable you to change vary greatly according to the type of element. For example, with a class Inspector, you can:

- Edit the class properties including name, stereotype, and superclass.
- Change the appearance of the class node on the diagram.
- Create and navigate hyperlinks between diagrams or diagram elements and other diagrams, elements on other diagrams, and external files or URLs.
- Add Javadoc comments such as @author and @version.
- Edit requirements information.
- Edit bean properties.

The property editors for the Inspector fields depend on the value types. Table 8 shows the icons that the Inspectors display for some kinds of fields.

Table 8 Inspector property editor icons

Icon	Name	Meaning
	(Asterisk)	Indicates a field that can take multiple values.
	File/Path Chooser	Opens a selection dialog box.
	Add/Remove Value	Opens a dialog box that displays the list of current property values and enables you to add or remove them.
	Edit	Enables you to add or modify textual values.
	Select Color	Opens a dialog box where you can choose foreground and background colors. Used in view adjustment fields.

To cancel or delete a new entry:

- Press **Esc** to cancel entries in text fields if you have not yet pressed Enter to complete your changes.
- If a value is already entered, use the **Undo/Redo** buttons on the main toolbar, or **Ctrl+Z**. This is valid for text fields, comboboxes, and checkboxes.
- To delete a value from a text field or a combobox, select it in the Inspector and press **Delete**.

Inspector tabset

The composition of the Inspector tabset changes depending on the selected element and where you made the selection. (For example, elements selected from the Explorer pane do not have *View* tabs.) This section describes the basic function of each tab of an ordinary class inspector, as shown in Figure 7.

In general, a tab displays two columns:

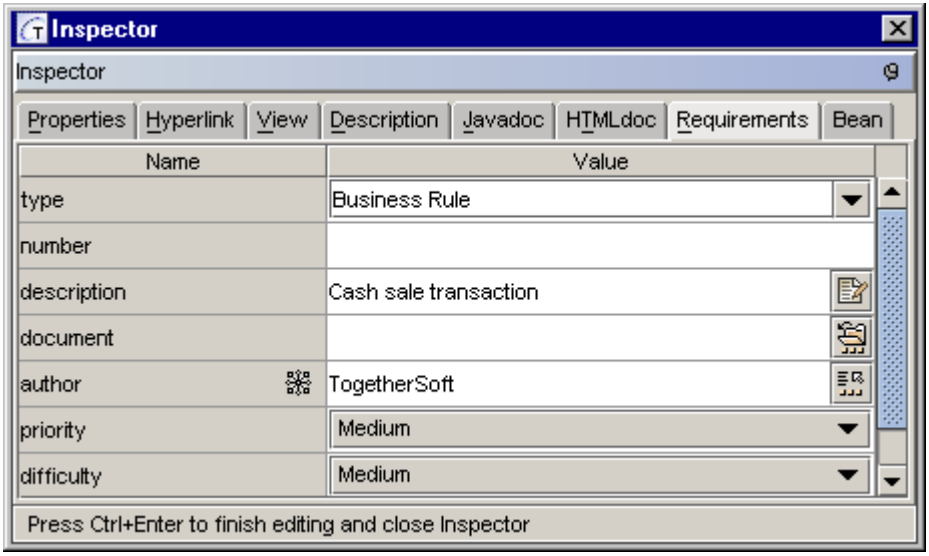
- **Name** shows the name of each property.
- **Value** displays the value or setting for the property.

To enter a value for string type properties, type a value in the edit field of the *Value* column. Some edit fields provide one of the following to help you quickly and accurately define a value:

- An arrow next to the field indicates a drop-down list of available values. You can select from the list or enter your own value in such fields.
- The asterisk symbol in the *Name* column indicates that the property can have multiple values (for example, *parameter* of an operation). Use commas to separate multiple values.
- A "browse" button next to the field indicates that an editor dialog is available. Multi-line string properties display a multi-line text editor dialog.
- A file chooser button next to the field indicates that the value must be the name of some object or element in the project. Click the button to open a dialog where you can select from available project elements.

Tip Use the keyboard shortcut `Alt+Insert` to edit the fields with file chooser buttons.

Figure 7 Properties Inspector for a class.



Properties tab

The Properties tab shows the properties of the currently selected element in the focus diagram or the diagram itself. You can view and modify values of properties.

Some properties can have multiple values. Multiple values are comma-delimited. Fields that enable multiple values are marked with an asterisk.

Hyperlink tab

The Hyperlink tab enables you to create hyperlinks to different types of artifacts and browse directly to them. You can create hyperlinks from the focus diagram or from a selected element on the diagram to:

- A diagram or diagram element anywhere in the project.
- A file that is external to your project.
- A URL on your company intranet or on the Internet.

You create, view, remove, and browse hyperlinks with the Hyperlink tab right-click menu.

View tab

The View tab enables you to set the foreground and background colors of an element selected in the Designer pane. When you start working with Together diagrams, the foreground and background colors are the default system colors.

To change the background color of a selected node:

1. Click the **Select Color** button in the background field.

2. In the resulting dialog box:
   - Choose the color from the standard Swing scale of colors using the Swatches tab, *or*
   - Define a custom color using the RGB (Red-Green-Blue) tab. In this case, you can choose the background color by moving sliders.
3. To restore the default color, click **Set Default**.

Javadoc tab

Inspectors for source-generating elements display the Javadoc tab. You can enter a description and specify values for Javadoc tags applicable to the selected element. These values are used when you generate Javadoc using the Documentation Generation feature.

Filling in the Javadoc fields automatically generates appropriate tags in the Javadoc tags in the source code.

- `@author` and `@see` tags allow multiple values. Click the file chooser buttons next to the fields to add values. In this case a separate tag is generated for each value in an Inspector field.
- Checking the box `@deprecated` creates an empty tag. Click the file chooser button next to the box to add a comment.

Requirements tab

You can track various requirements properties including type, priority, and difficulty for diagrams and individual elements. You can specify a requirements document for the diagram or elements.

Note A hyperlink to the document is not created when you specify the requirements document. Use the Hyperlinks tab to create such links.

Beantab

This tab is added to the Inspector when *Recognize JavaBeans* options are checked on. For classes, this tab displays JavaBean/EJB properties and events on the following lower tabs:

- General: general JavaBean attributes.
- Properties: bean properties such as getter, setter, bound, and constrained.
- Events: bean event sets.

To turn on Recognize JavaBeans:

1. Choose **Tools | Options | <level>** from the main menu.
2. Expand the *View Management* node on the left.
3. Choose *JavaBeans*.

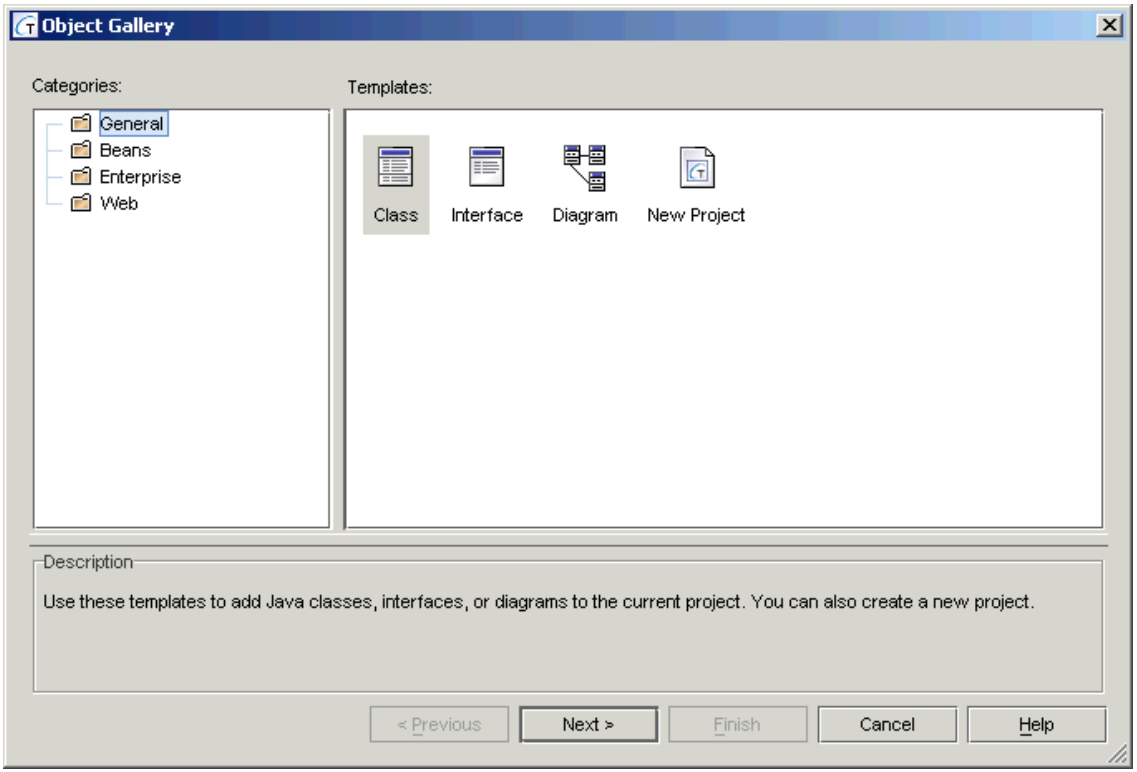
4. Check the properties listed on the right side that are appropriate for your project.
5. Click OK to save and quit the dialog.

Object Gallery

The Object Gallery is a collection of "templates" for creating new files in a project. You choose the type of object you want to create, and specify the location, name, and relevant properties. This creates a skeleton file for you to work with in the current project.

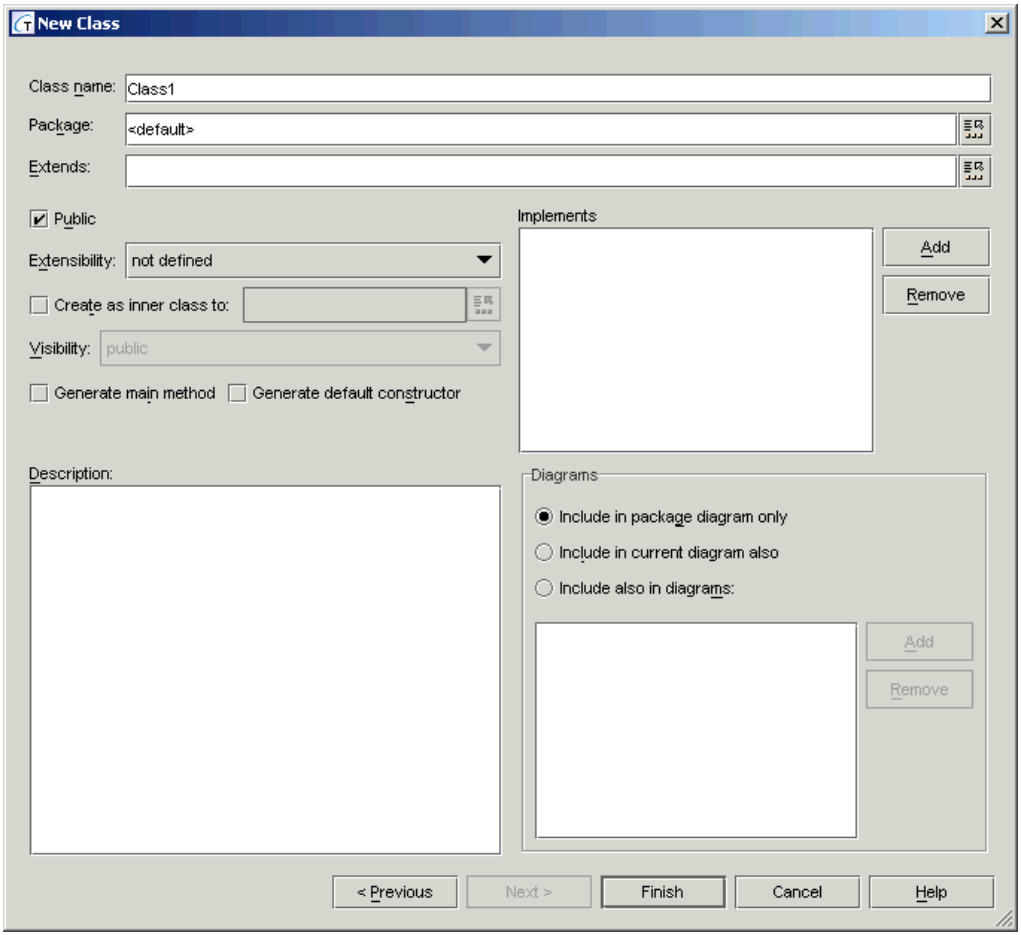
Use the Object Gallery to create diagrams, JavaBeans, and e-commerce objects (EJBs, enterprise and web applications, JSPs, servlets, and so on). The types of objects available in the Object Gallery depend on which features are activated. For example, Figure 8 shows the Object Gallery when the E-Commerce feature is activated.

Figure 8 Object Gallery with the *General* category selected.



When you choose an object to create, the next page displays object properties for you to define. Clicking the Finish button creates the skeleton for the object, with the specified properties. Figure 9 shows the page for creating a new Java class.

Figure 9 Specifying properties for a Java class in the Object Gallery.



To work with the Object Gallery:

1. Choose **File** | **New** to open the Object Gallery. Alternatively, click the New button on the main toolbar or press `Ctrl+N`.
2. Under *Categories* on the left side, select a category of objects to choose from.
3. Under *Templates* on the right side, select a type of object to create.
4. Click the **Next** button or double-click the object icon to go to the next page.

5. Define the object properties as desired. If the Next button is still enabled, click it to access more properties.

   For help defining properties, click the **Help** button. This opens the Object Gallery topic in the online help, where you can find detailed documentation on properties for specific types of objects.

6. Click **Finish** to accept the values and create the object.

Chapter

2

Setting Your Personal Preferences

This chapter explains how to set-up preferences for workspaces and view management. This chapter includes the following topics:

- "Setting up workspaces" on page 49.
- "Using view management" on page 50.
- "Setting JBuilder Integration options" on page 53.

Setting up workspaces

Together enables you to define, save, and recall different arrangements of main window panes. These saved, reusable pane arrangements are called *workspaces*.

Accessing different workspaces

Workspaces are project-specific. There are two workspaces available in Together:

- *default workspace:* This workspace contains the Designer, Explorer, and Message panes. Every project has a default workspace. Together names the default workspace with the same name as the project.
- *design workspace:* This workspace contains the Designer pane and Inspector.

You can access the different workspaces from the right-most item on the main toolbar. The dropdown list shows the workspaces available for the current project.

Managing workspaces

You can create new workspaces, delete workspaces, and rename workspaces.

To create a new workspace:

1. Arrange the main window and panes for the new workspace layout.
2. Choose **View | Workspaces | Create Workspace** from the main menu.
3. Enter a name for the new workspace in the resulting dialog box.
4. Click **Ok** to quit the dialog and save the result.

To rename or delete an existing workspace:

1. Choose **View | Workspaces | Manage Workspaces** from the main menu.
2. Choose the workspace that you want to rename or delete from the list in the resulting dialog.
3. To delete the workspace, click **Remove**. To rename the workspace, enter the new name in the *Workspace name* field.
4. Click **Ok** to quit the dialog and complete the action.

Note You cannot delete the default workspace. You can, however, rename it.

Saving workspaces

You can rearrange the panes and window layout for any workspace. Whether Together saves a rearrangement depends on the workspace save option.

To set the save workspace option:

1. Choose **Tools | Options | Default Level** from the main menu.
2. Choose *General* from the option list, but do not expand it.
3. In the right frame, check *Save workspace before leaving*.
4. Click **OK** to close the dialog and save the settings.

With the save option checked, the current state of a workspace is automatically saved when you leave it. If the option is not checked, any changes to the workspace are lost when you leave it. If the save workspace option is turned off, you can still manually save the workshop. **Select View | Workspaces | Save Workspace** from the main menu before leaving.

Using view management

The Together view management feature allows you to control the type of data displayed in different views of a model. Thus, different members of a project team can view aspects of the model that are relevant to their roles. For example, if you are the domain expert of a project, you can set up a model so that you do not see implementation details, or other information irrelevant to your tasks.

Controlling the level of detail shown in diagrams

To control the desired level of detail within a diagram, follow these steps:

1. From the **Tools** menu, choose **Options** | **Default Level** (you can also choose **Project Level** or **Diagram Level**).
2. From the tree, choose *View Management*.
3. For *Diagram detail level*, choose *Default*, *Analysis*, *Design*, or *Implementation*.

Controlling how members display in diagrams

To control whether members of a class display in UML format or a corresponding language, follow these steps:

1. From the **Tools** menu, choose **Options** | **Default Level** (you can also choose **Project Level** or **Diagram Level**).
2. From the tree choose *View Management* to see the names and values of its options.
3. For *Member format*, select *UML* or *Language*.

Showing and hiding subpackage contents in diagrams

Check this option to display subpackage content as package icons with the lists of classes, interfaces, and underlying subpackages. (In the Options dialog, expand *View Management* and choose *Show subpackage contents.*)

To show or hide subpackage contents, follow these steps:

1. From the **Tools** menu, choose **Options** | **Default Level** (you can also choose **Project Level** or **Diagram Level**).
2. Expand *View Management*, and select *Show subpackage contents.*
3. Check the box for *Show subpackage contents* to show contents. Uncheck the box to hide contents.

Controlling how JavaBean properties display in class diagrams

To control how JavaBean properties display in class diagrams, follow these steps:

1. From the **Tools** menu, choose **Options** | **Default Level** (or **Project Level**).
2. From the tree, expand *View Management*, then choose *JavaBeans Properties.*
3. Check the boxes for *Recognize JavaBeans.*

Note This option appears for the default and project levels only. It does not appear under the Diagram Level options.

If the option *Recognize JavaBeans* is checked, the Bean tab appears on the Object Inspector of the classes, where you can add bean properties, getters and setters, and event sets. The *Show attributes and accessors* option controls whether bean properties and events show up on the class icon.

Showing and hiding referenced classes in diagrams

To show or hide referenced classes in a diagram, follow these steps:

1. From the **Tools** menu, choose **Options** | **Default Level** (you can also choose **Project Level** or **Diagram Level**).
2. From the tree, choose *View Management*.
3. For *Referenced classes*, select *Show name* or *Hide*.

Showing dependencies between classes and interfaces

To show dependencies between classes and interfaces within a diagram, follow these steps:

1. From the **Tools** menu, choose **Options** | **Default Level** (you can also choose **Project Level** or **Diagram Level**).
2. From the tree, expand *View Management*, and choose *Dependencies.*
3. For *Check*, select *Declarations* only or *All usages.*

Note Recognizing dependencies can result in slow performance and large diagrams.

Controlling the display of sequence diagrams

To access the sequence diagram options for view management, follow these steps:

1. From the **Tools** menu, choose **Options** | **Default Level** (you can also choose **Project Level** or **Diagram Level**).
2. From the tree, expand *View Management*, and choose *Sequence diagram.* You can set options for this node. You can also expand *Sequence diagram* to view more options.

Banning destinations in class diagrams

You can use the *Banned Destinations* option to keep association links under control. If the links to standard classes are not filtered from view, your diagrams can become crowded with links that impede comprehension. Some standard Java classes are banned by default.

To ban destinations in class diagrams, follow these steps:

1. From the **Tools** menu, choose **Options** | **Default Level** (or **Project Level**).

2. From the tree, select *Banned Destinations,* and enter the class that you want to ban.

In addition to the fields provided in the options dialog, you can add more fields by editing the `$TGH$/config/viewManagement.config` file. For more information, refer to the online Description field that appears for banned destinations in the Options dialog.

Showing or hiding aggregations of diagram and EJB elements

The view management *Show* and *Hide* options allow you to control the presentation of information in diagrams. *Together* comes with a number of predefined view management options and provides additional ones that you can custom-define.

To access show and hide options for view management, follow these steps:

1. From the **Tools** menu, choose **Options** | **Default Level** (you can also choose **Project Level** or **Diagram Level**).
2. From the tree, expand *View Management*, and then expand *Show* (or *Hide*).

Setting JBuilder Integration options

JBuilder *Integration* options include the JBuilder installation directory, project synchronization, and options for displaying confirmation and information dialogs.

To access these options:

1. From the Together main menu, choose **Options** | **Default Level** (you can also choose **Project Level** for the options specifically noted).
2. Select the JBuilder *Integration* node on the left of the dialog.

The following options display on the right of the dialog:

- JBuilder *installation directory:* This option specifies the root folder for your JBuilder installation. This setting was made during Together installation, but if you have changed the location of your JBuilder installation you will need to set the new location here.
- JBuilder *project synchronization:* This option determines how changes to project properties in JBuilder (such as adding source paths, and so on) are synchronized in Together. Note that changes to sources are always synchronized automatically. (This option is also available at the **Project Level**.)
    - **Synchronize project with** JBuilder **automatically:** Project changes in JBuilder are automatically synchronized when opening a project in Together or switching the focus to Together, without asking for confirmation.

- **Synchronize project with** JBuilder **on user request:** When opening a project in Together or switching the focus to Together after making changes in JBuilder, a confirmation dialog asks whether you want to synchronize changes to the project. In each instance, you may choose to synchronize or not.
- **Never synchronize projects in future:** Changes to project properties are never synchronized automatically and you are never prompted to synchronize changes.

- *Show message on opening JBuilder project:* If this option is checked, an information dialog opens when a JBuilder project is opening. When you open a JBuilder project in *Together*, a Together project with the same name and properties is created if it does not already exist.
- *Confirm starting* JBuilder *on edit:* Enabling this option displays a confirmation dialog to start JBuilder upon choosing the **Edit** command from the class element right-click menu. (This option is also available at the **Project Level**.)

Note: You can also set the JBuilder *project synchronization* property at the project level by accessing Together project properties. Choose **Project | Project Properties** from Together's main menu, and click the drop down arrow to choose a synchronization method.

Chapter

3

Configuring Options

This chapter explains how to configure options for *your* installation of Together. If you are the Together administrator and need to configure options on a more global level that impacts all users, see "Configuring Together for Multiple Users" on page 481.

This chapter includes the following topics:

- "Activating and deactivating modules" on page 55.
- "Overview of configuration levels" on page 56.
- "Setting options in default mode" on page 56.
- "Setting options in advanced mode" on page 58.
- "Viewing and editing options" on page 60.
- "Reference guide to options" on page 61.
- "Common Configuration Tasks" on page 66.
- "Configuring Together for improved performance" on page 69

Activating and deactivating modules

Some Together features need to be activated before you can use them. This also applies to most integrations for third-party products such as Versant and Persistence Powertier.

Note By activating a module, you can also view options specific to the module. Similarly, deactivating a module removes the options specific to the module from the user interface.

To activate (or deactivate) a module, follow these steps:

1. From the **Tools** menu, choose **Activate/Deactivate Features**.
2. Select the **Together Features** tab (or the **Integrations** tab).
3. Check the modules that you want to activate; uncheck the modules that you do not need to use. For your convenience, each module's path name and size is provided.
4. Click **OK.**

Depending on the number of modules you have activated, it may take several seconds for Together to refresh the user interface.

Overview of configuration levels

Set options for each of the three levels of Together's multilevel configuration:

- **Default level**: Settings apply to the entire **Together** installation, unless overridden at the project or diagram levels
- **Project level**: Settings apply only when a project is open, unless overridden at the diagram level
- **Diagram level**: Settings apply to a specific diagram (specifically, the diagram that is open at the time the option is set)

For any of the three levels, set options using a dialog in either *default* or *advanced* mode. Default mode allows you to set options for a single configuration level. Advanced mode allows you to set options at multiple levels without re-invoking the dialog. For more information see, "Setting options in advanced mode" on page 58.

Together allows you to apply settings for most of the options at any level. Consequently, options cannot be overridden at a lower (that is, more local) level.

When Together is installed as a server-based application, options are centralized and shared by all users. The Together administrator can customize the configuration so that specific settings apply across the enterprise (Default level marked final), or globally for a team (Project level marked final). To learn how to configure Together for a shared installation used by multiple users, see "Configuring Together for Multiple Users" on page 481.

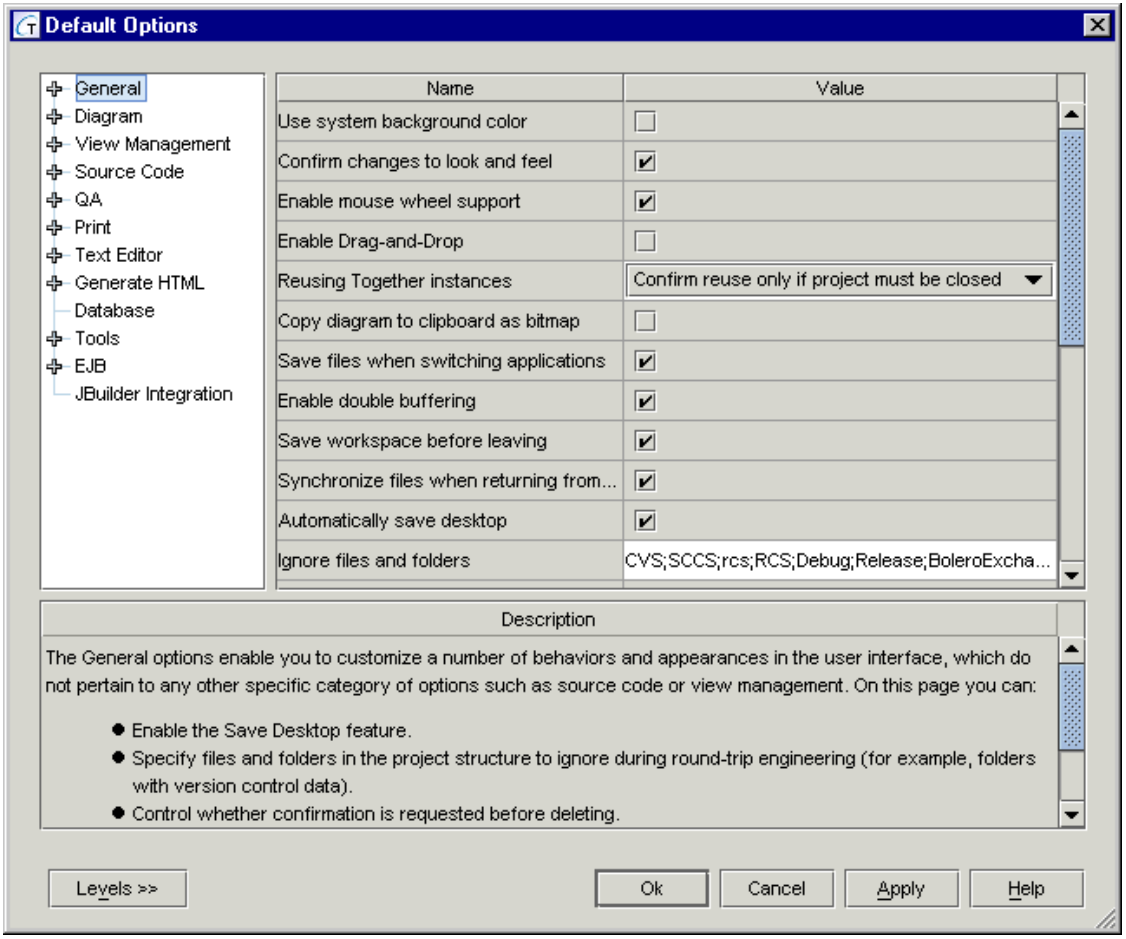
Setting options in default mode

Use the options dialog in default mode to set options at a specific default, project, or diagram level. (In contrast, advanced mode allows you to set options at more than one level.)

To set options in default mode, follow these steps:

1. From the Tools menu, select **Options**
2. Select **Default Level**, **Project Level**, or **Diagram Level**. The dialog for the respective level appears. For example, Figure 10 shows the Default Options dialog.
3. Set the options as needed. For more information, see "Viewing and editing options" on page 60. For a reference to the options, see "Reference guide to options" on page 61.
4. Click **OK** to save settings.

Figure 10 Default Options dialog in default mode


Default Options
General
Diagram
View Management
Source Code
QA
Print
Text Editor
Generate HTML
Database
Tools
EJB
JBuilder Integration
Name
Value
Use system background color
Confirm changes to look and feel
Enable mouse wheel support
Enable Drag-and-Drop
Reusing Together instances
Confirm reuse only if project must be closed
Copy diagram to clipboard as bitmap
Save files when switching applications
Enable double buffering
Save workspace before leaving
Synchronize files when returning from...
Automatically save desktop
Ignore files and folders
CVS;SCCS;rcs;RCS;Debug;Release;BoleroExcha...
Description
The General options enable you to customize a number of behaviors and appearances in the user interface, which do not pertain to any other specific category of options such as source code or view management. On this page you can:
Enable the Save Desktop feature.
Specify files and folders in the project structure to ignore during round-trip engineering (for example, folders with version control data).
Control whether confirmation is requested before deleting.
Levels >>
Ok
Cancel
Apply
Help

Setting options in advanced mode

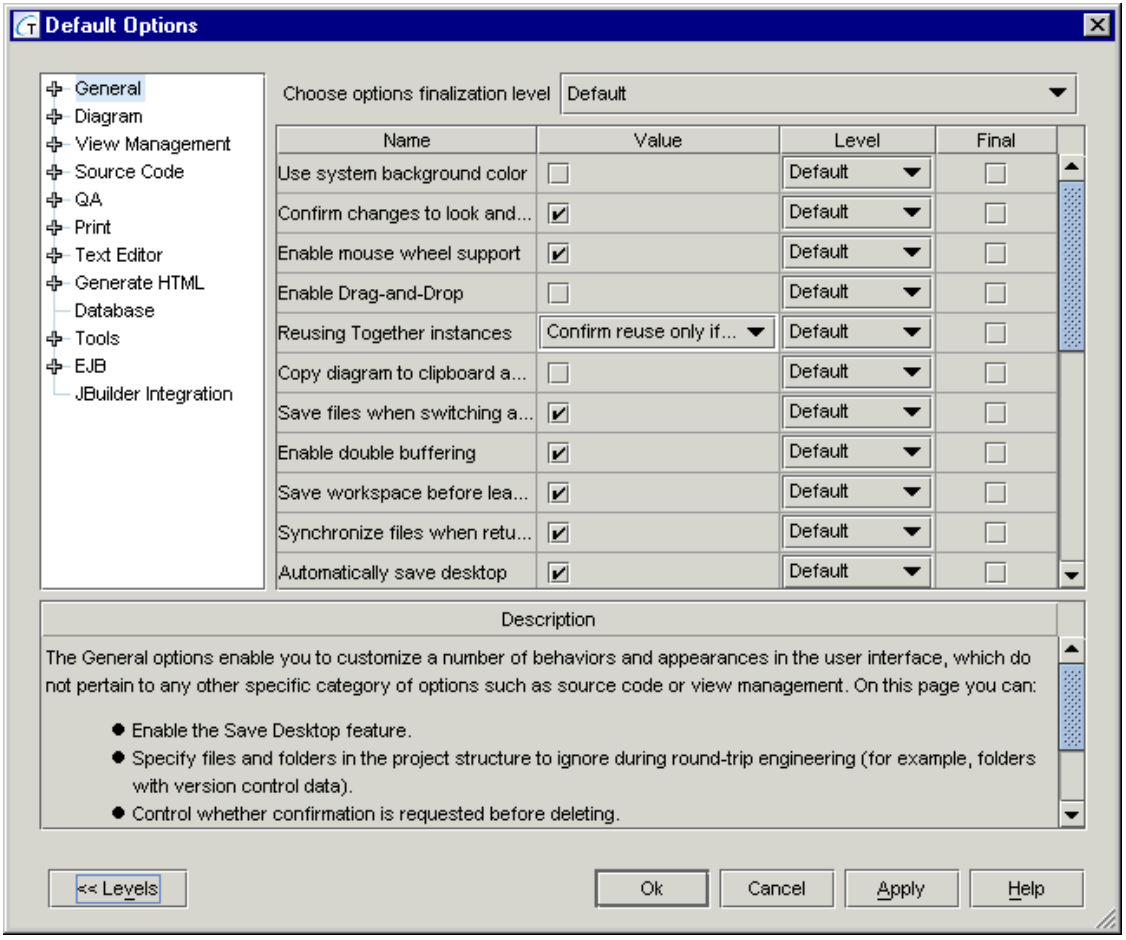
Use the options dialog in advanced mode to set options at multiple levels without re-invoking the dialog. The options dialog for each level (default, project, diagram) includes a *Levels* button that you use to toggle between default and advanced modes.

In contrast to default mode, advanced mode includes the columns *Level* and *Final*, as shown in Figure 10. The Level column allows you to select the configuration level. In this case, Level indicates the level at which each configuration option is currently set, assuming the default level definitions:

Default = Option is set at the Default level
Project = Option is set at the Project level
Diagram = Option is set at the Diagram level

The Final column indicates that an option cannot be overridden at a level more local than the one shown in the Level column. Change any settings that are not marked final at a “higher” (that is, more global) level. In a local installation, you typically have complete control over your configuration and the settings at all levels. In a shared installation, change only those settings not marked as final by the system administrator in the shared configuration.

Figure 11 The Default Options dialog in advanced mode



To set options in advanced mode, follow these steps:

1. Open a project. To set options for a specific diagram, select the diagram.
2. Open the Options dialog for the lowest available level by choosing **Tools** | **Options** | ***available_level*** .
3. Click the **Levels>>** button to enter advanced mode.
4. Set options for each level. For more information, see "Viewing and editing options" on page 60. For a reference to the options, see "Reference guide to options" on page 61.
5. Click **Apply** to save changes as you work.
6. Click **OK** to save changes and close the dialog.

Viewing and editing options

The information in this section assumes that you have accessed the options dialog. For more information, see "Setting options in default mode" on page 56 or "Setting options in advanced mode" on page 58.

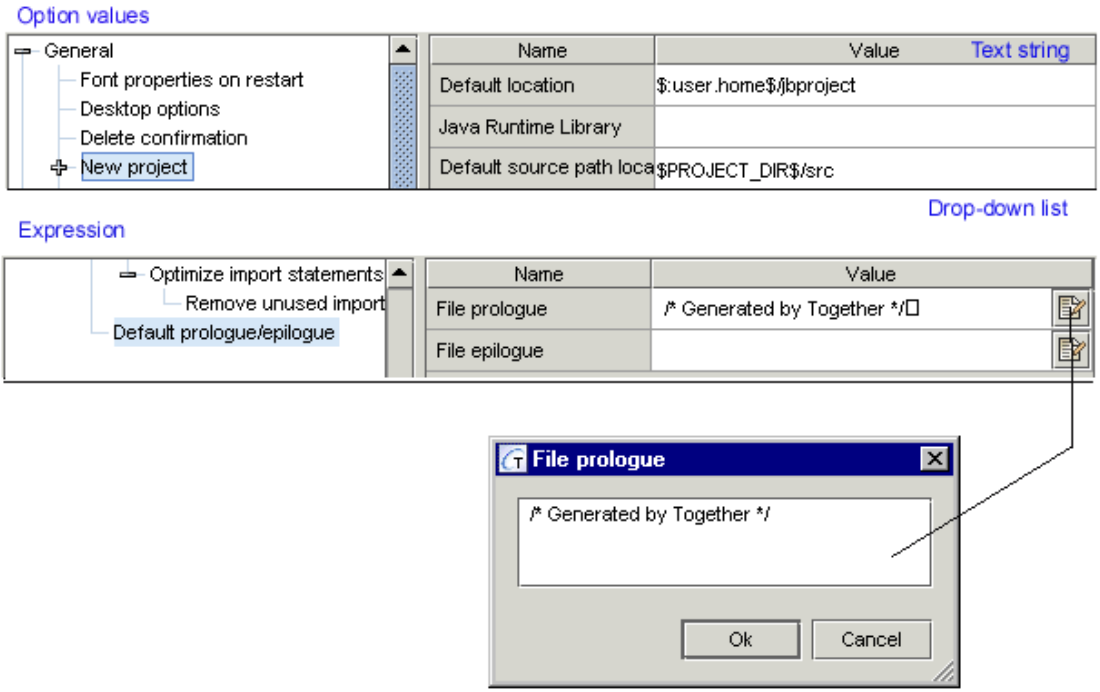
The Options dialog logically groups the options as nodes in a tree. When an option consists of groups of sub-options, you can expand the node to show all available sub-options.

Often options can be set for a feature at *each* hierarchical level of the tree view. Select a node, regardless of its hierarchical level, to view the values that can be set. For example, while the following nodes appear in different hierarchal levels of the tree, each node provides values that can be set for a sequence diagram:

- View Management
- Sequence diagram
- Generate Sequence Diagram

Options have editors depending on the type of value. If an option has multiple values, the values are comma-delimited. Figure 12 illustrates these concepts.

Figure 12 Examples of values for options



Values indicated by checkboxes

Together uses checkboxes to specify a Boolean value for an option. A checked box means “true” or “yes”, and an unchecked box means “false” or “no”.

Accessing context-sensitive help for nodes and options

Help for each node and option displays directly in the dialog under Description. To see a general description of a group of options, click on its node. To see the description of an individual option, click its name.

Resizing the Options dialog

The Options dialog box can be resized. To resize the width of the Name and Value columns drag the separator between the column headings. In addition, the size of the Description area can be changed.

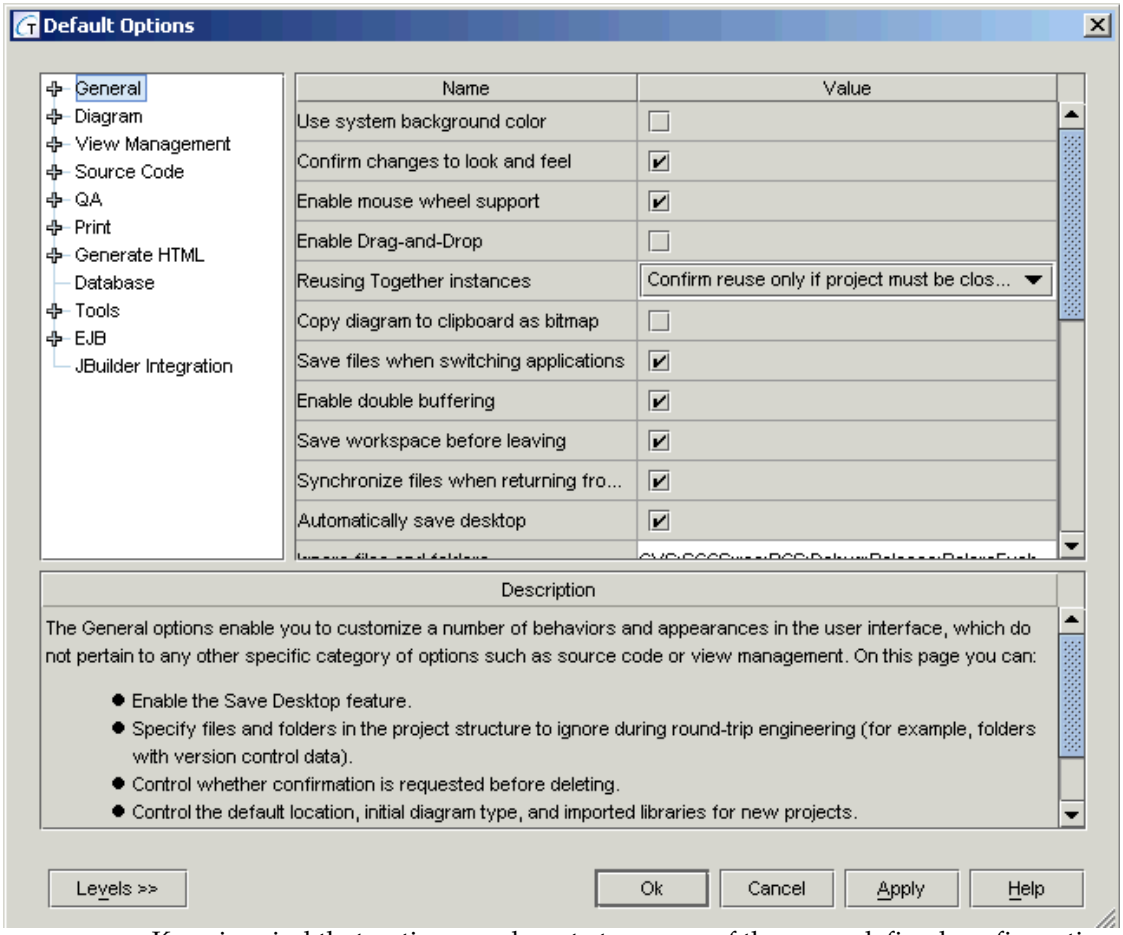
Encoding of the configuration options

Starting with the version 6.0.1, configuration files of Together are written using UTF-8 encoding. It is strongly advisable to edit configuration files via the Options dialog. However, if a configuration file needs to be edited externally, use an editor that supports UTF-8 encoding (for example, Windows 2000 Notepad), and save the file accordingly.

Reference guide to options

This section summarizes the options that can configured in Together. Options are listed and grouped within the tree of the Default Options dialog (as well as the Project Options and Diagram Options dialogs), as shown in Figure 13. Help text for the nodes and the individual options appears in the Description field of the options dialog.

Figure 13 Tree listing options and option groups


Default Options
General
Diagram
View Management
Source Code
QA
Print
Generate HTML
Database
Tools
EJB
JBuilder Integration
Name
Value
Use system background color
Confirm changes to look and feel
Enable mouse wheel support
Enable Drag-and-Drop
Reusing Together instances
Confirm reuse only if project must be clos...
Copy diagram to clipboard as bitmap
Save files when switching applications
Enable double buffering
Save workspace before leaving
Synchronize files when returning fro...
Automatically save desktop
Description
The General options enable you to customize a number of behaviors and appearances in the user interface, which do not pertain to any other specific category of options such as source code or view management. On this page you can:
Enable the Save Desktop feature.
Specify files and folders in the project structure to ignore during round-trip engineering (for example, folders with version control data).
Control whether confirmation is requested before deleting.
Control the default location, initial diagram type, and imported libraries for new projects.
Levels >>
Ok
Cancel
Apply
Help

Keep in mind that options can be set at any one of three pre-defined configuration levels. Know what level you are working on before changing any option settings.

Default and project level options

Table 9 lists the options that you can set at the default and project levels. The table lists the options in the order they appear in the Default Options dialog.

Note If the user interface does not show the options listed for a specific feature, make sure that the feature is activated. Some features must be activated before using them. See "Activating and deactivating modules" on page 55 for more information.

Table 9 Default and project level options

Option	Allows you to...	On this node you can...

Table 9 Default and project level options (continued)

General	Customize a number of behaviors and appearances in the user interface.	• Control background color and font properties. • Set to auto-synchronize files when returning from other applications (such as an external editor or IDE). • Enable or disable the Saved Desktop feature that remembers your desktop settings between sessions. • Specify files and folders in the project structure to ignore during round-trip engineering. • Control the display of delete confirmations. • Control the default location, initial diagram type, and referenced libraries for new projects. • Control whether the Message pane opens on errors. • Enable or disable email exception reports to the Together development team. • Control whether Together will search archives for diagram files.
Diagram	Control a number of default behaviors and display properties that apply only to diagrams.	• Control the default link routing method. • Control the alignment, layout, justification, and initial maximum width of classes in diagrams. • Change the font used in diagrams. • Specify how association links are drawn and how and whether they are represented in classes. • Specify whether to generate metafile images of diagrams when saving diagrams. • Control the display of the background grid in the Designer pane.

Table 9 Default and project level options (continued)

View Management	Control what you want to see and when. Specify how different kinds of elements display in the diagrams, or even whether they show up at all. For more information, see "Using view management" on page 50.	• Control the general level of detail shown in diagrams. • Control whether members in classes display in UML or Java format. • Show or hide subpackage contents in diagrams. • Control how JavaBean classes display in Class diagrams. • Show or hide referenced classes in diagrams. • Control the display of dependencies. • Show object class names and message numbers in Sequence diagrams. • Control the display of messages in Sequence diagrams. • Set banned destinations. • Specify the maximum call stack depth for Sequence diagrams generated from operations. • Show or hide aggregations of diagram elements and EJB elements.
Source Code	Control a number of default behaviors and appearances that apply to the formatting of source code during forward and reverse engineering operations.	• Specify the relative position of Attribute declarations and Operation declarations within Class declarations (Attributes first or Operations first). • Control how link attributes are handled when the destination is deleted. • Specify exactly how source code and comments are formatted (in-line breaks, space preservation, separators, and so on). Includes Javadoc comment formatting options for comments. • Customize source file prologue and epilogue text (the "Generated by" text at the head of source code files).
QA	Customize QA options. **Note:** To view this option, you need to activate the QA module.	• Specify the path to saved sets of audits and metrics. • Define the applicable scope of Quality Assurance.

Table 9 Default and project level options (continued)

Print	Set a number of defaults that apply to printing diagrams and generated documentation.	• Set default paper size or define a custom one (for example, for printing on a plotter). • Set page orientation and margin sizes. • Set a number of other print options such as print zoom level, page border/footer, and so on.
Generate HTML	Control the inclusion/exclusion of various content in the output of the standard HTML documentation generation facility (Tools | Generate HTML).	• Include or exclude author and version tags in generated output. • Specify all Javadoc settings. • Specify which visibility levels of classes to include in generated HTML output. **Tip:** Set these options as you begin the documentation generation process. From the Generate HTML dialog, click Options to display the Options dialog with only the Generate HTML node visible. If you change any settings, they are used for the current documentation generation operation only and then discarded. Option settings for your configuration are not changed.
Database	Customize a number of common database properties that are used in operations with databases.	• Define the amount of time that Together will wait for a connection with DBMS. • Control whether Together replaces tables after generating a DDL. • Specify whether to use quoting symbols for identifiers in the resulting DDL.
Tools	Edit various fields to point to appropriate file-based development tools (such as compilers, debuggers, IDEs, and editors). The Tools options contain the tool definitions for your configuration. The External editor tool is pre-defined for you.	
EJB	Set J2EE defaults	• Select the application server for J2EE deployment, which also determines which EJB specification new EJBs will be designed to meet. • Edit EJB suffixes that are used for EJB recognition.

Table 9 Default and project level options (continued)

JBuilder Integration	Use these options to determine how JBuilder projects will be handled in Together.	• Specify the root folder for your JBuilder installation. This setting was initially made during Together's installation, but if you have changed the location of your JBuilder installation you will need to set the new location here. • Determine how changes to project properties in JBuilder (such as adding source paths, and so on) are synchronized in the Together project file. • Specify settings to show an information dialog whenever a JBuilder project is opening. • Control whether or not to start JBuilder automatically upon choosing to edit a source code file.

Diagram level options

Set the following options at the diagram level:

- Diagram
- View Management
- Print

These options can also be set at the default and project levels. See Table 9 for a description of each.

In addition, when activating IDL Export Support, you can set diagram level options. For more information, see "Activating and deactivating modules" on page 55.

Common Configuration Tasks

This section groups typical configuration tasks.

How to make Association links display a directional arrow

Configure the default properties of Association links in the Options dialog, and the properties of individual links in the properties Inspector of the link.

To configure default Association properties, follow these steps:

1. Choose **Tools** | **Options** | **Default Level** to launch the Options dialog.
2. If necessary, click the Levels button to enter Advanced mode to set options at multiple configuration levels.

3. From the tree-view, expand **Diagram** and select **Associations**.
4. Set the value of *Draw directed* as required:
   - Automatic - Links represented by the attributes whose names start with "link," will be shown directed. All other links will be shown undirected.
   - All - All links will be shown directed.
   - None - All links will be shown undirected.
5. Click **Apply** and then **Ok**.

This option applies only to Association links whose "directed" property in the link's Inspector is set to "Automatic". If it is set to "Directed" or "Undirected," the link will always be displayed according to that setting, regardless of the value set for this option.

Tip You can also set *Show as attributes* so that attributes which are displayed as links show in the attributes section of classes.

How to change the default source file header for the generated code

1. Choose **Tools** | **Options** | **Default Level** to launch the Options dialog.
2. If necessary, click the Levels button to enter Advanced mode to set options at multiple configuration levels.
3. From the tree-view, expand **Source Code** and then expand the **Java** node.
4. Choose *Default prologue/epilogue*.
5. In the Value column, edit the default text as required. For example: "Generated for XYZ. Corp. Copyright (c)2001. Company confidential."
6. Click **Apply** and then **Ok**.

How to configure Stereotypes

Customizing stereotypes is a low-level configuration task and requires Java programming. This enables you to populate stereotype lists, modify default stereotype values, and specify RGB color values for stereotypes, and so on. See "How to configure Stereotypes" on page 67 for more information.

How to customize Inspector properties

For more information on how to modify the property names and/or default values in properties Inspectors, add your own properties, or delete properties, see "How to customize Inspector properties" on page 67. Customizing inspector properties is a low-level configuration task and requires Java programming.

How to hide and show elements

1. Choose **Tools** | **Options** | ***Applicable_Level*** to launch the Options dialog.

2. If necessary, click the Levels button to enter Advanced mode to set options at multiple configuration levels.
3. From the tree-view, expand **View Management**.
4. Expand **Show**. To hide the elements defined by one of the options (Inheritance links, for example), clear the option checkbox. To re-show elided elements, check the option box.
5. Click **Apply** and then **Ok**.

See "Using view management" on page 50 for more information.

Note that individual elements can be hidden in diagrams using the Diagram right-click menu. If you do not see an element in a diagram, and the element is not hidden by View Management options, choose Show Hidden from the Diagram right-click menu and check the hidden elements list.

Define filtering expressions for the User-Defined options under the Show options of the View Management page. Study the expressions in the pre-defined Show options to learn how to show or hide different elements.

The other options on the View Management page of the Options dialog may hide some kinds of information. For example, *Diagram Detail Level* can hide visibility symbols.

How to enable mouse-wheel support (Windows only)

Java VM 1.2, 1.3 does not support mouse wheel events. To scroll a diagram or a frame when using a mouse equipped with a wheel, Together usually transforms mouse wheel events into Ctrl-Up, Ctrl-Down, Up or Down keystrokes.

To enable mouse-wheel support:

1. On the main menu, choose **Tools** | **Options** | ***Applicable_Level***.
2. From the tree-view, choose **General**.
3. Check or uncheck *Enable mouse wheel support*.

   If the option is checked, this feature is enabled. If unchecked, Together does not transform mouse wheel events.
4. Click **Apply** and then **Ok**.

Note This option applies only when Together is running with the Sun Java Virtual Machine (JVM) under Windows. Unlike most options, this one requires restart of Together before taking effect.

Configuring Together for improved performance

One way to improve performance is to turn off unnecessary features in the Activate/Deactivate Features dialog (see "Activating and deactivating modules" on page 55). This should make the project load faster and improve performance in general.

In addition, many of the config options affect the performance of Together. To improve overall performance, consider the following tips:

- Uncheck *General: Save files when switching applications* and/or *General: Synchronize files when returning from other applications*. If both options are turned on (checked), task switching is slower. Turn off one or both to improve performance.
- Uncheck *General: Enable Drag-and-Drop* (this is the default setting). The drag-and-drop behavior varies depending on your operating system and whether other performance-sensitive options are enabled, so try turning this option on and off to see whether there are any noticeable differences in performance on your system.
- Uncheck *Diagram: Update link sources when destination changes* (this is the default setting). For each link destination, there could potentially be multiple link sources to check and update. In any case, there is always more updating to process if this option is checked.
- Uncheck *Diagram: Antialias graphics* (this is the default setting). If this option is checked, it slows performance when working with diagrams.
- Uncheck *View Management - JavaBeans: Recognize JavaBeans.*
- In *View Management - Dependencies*, set *Find between classes* to *None* (this is the default setting). If you need to set this option to *Diagram local* or *All*, set the *Check* option to *Declarations only* to improve performance. In addition, unchecking *Recognize @see tags as hyperlinks* slightly improves performance.

Also consider the following options, which do not significantly affect overall performance but could make a difference, depending on your specific configuration or tasks:

- *General: Ignore files and folders*. The more files and folders included in the ignore list, the faster project sources can be processed.
- *General: Create design elements as standalone*. A large number of standalone design elements in a project may negatively impact performance. Consider turning off this option if you anticipate creating many design elements. If you will not create a lot of design elements, this option will not affect performance.
- *Diagram: Generate metafiles when saving diagrams*, *Diagram: Show controls for compartments*, and *Diagram - Grid: Show grid*. These options have a minor impact on performance when working with diagrams.

- *View Management - Show* options. The performance impact of these filters depends on the total number of options shown (checked). The more options hidden (unchecked), the better performance. Shown *Dependencies* and *Hyperlinks* have the most negative impact on performance.

The following tips apply to the performance of specific tasks:

- **Generating sequence diagrams.** In the option *View Management - Sequence diagram - Generate Sequence Diagram Options: Depth of call nesting*, set a low value (1 - 3) for the fastest parsing speed. In the same group of options, unchecking *Show full diagnostic* also slightly improves the speed of generating sequence diagrams.
- **Audits and metrics.** Uncheck *QA - Process classes: Process classes from Search/ Classpath* for significantly faster audits and metrics processing (this is the default setting).
- **Generating HTML documentation.** Uncheck *Generate HTML: Update package dependencies* to generate documentation faster (this is the default setting).

Part

2

CREATING AND MANAGING PROJECTS

- Chapter 4, “Working with Projects”
- Chapter 5, “Reverse Engineering”
- Chapter 6, “Importing and Exporting Information”

Chapter

4

Working with Projects

Almost all the work in Together is done in the context of projects. This chapter explains how to create and edit project settings in Together. It includes the following topics:

- "Understanding Project Basics" on page 73.
- "Creating new projects" on page 75.
- "Editing project properties" on page 76.
- "Running multiple project instances" on page 82.
- "Setting up large projects" on page 83

Important To use projects in both Together Edition for JBuilder and JBuilder, you must refer to *Getting Started with* Together Edition for JBuilder for information on setting up a new project and working with existing projects. You must follow the instructions in *Getting Started* to synchronize your project between Together and JBuilder. To view the manual, choose **Help | Manuals | Getting Started** from Together' s main menu.

Understanding Project Basics

To begin modeling with Together, you need to create project.

At minimum, a project consists of the following:

- Primary root directory
- Project file
- Default package diagram

Primary root directory

The primary root directory stores the project file and any initial diagrams created with the project. It also serves as a repository for project-level properties files. (Project level is one of the pre-defined configuration levels in Together as described in "Overview of configuration levels" on page 56).

When creating a project, select a new or existing directory for the primary root directory. By default, Together synchronizes any source code it finds in the primary root directory and any subdirectories below it.

Important Specifying the top-level directory of a particularly large code base as the primary root directory can slow the performance of reverse engineering. To avoid this, redefine the parsing of project resources. For more information, see "Performance tuning" on page 84.

The scope of a project is not limited to a single root directory. Instead, you can specify multiple directories as project root directories, include or exclude subdirectories of any root, and exert some initial control over how these directories are treated during synchronization with source code.

Project file

Together recognizes both Together and JBuilder project files in the Explorer pane. The Explorer pane, uses the JBuilder icon and the Together icon to indicate project files. The file name of a JBuilder project has the `.jpx` extension, and the file name of a Together project has a .tpr extension. For more information on using JBuilder with Together, refer to the *Getting Started Guide* located in the `$TGH$/doc` directory.

The <default> diagram

When creating a new project, Together generates a diagram that illustrates the physical project content contained in the primary root package. The generated diagram is named `default` and displays the default diagram icon. The name of the underlying diagram file is `default.dfPackage`. The `default` diagram shows Package icons representing subdirectories of the primary root directory as well as the classes (and other elements) of any source code files found in the primary root.

When creating a new project, opt to specify an initial diagram that Together creates along with the project. Class diagram is the default type, but you can specify another type of diagram as the initial diagram, or *none* for no initial diagram. Together always generates the `default` diagram.

Note that there is no default diagram that opens every time you open a project. Any diagrams that open along with a project are controlled by the configuration settings under Desktop Options (see Chapter 3, "Configuring Options"). If properly set, Together remembers which diagrams were open when the project closed and reopens the diagrams the next time you open the project.

Creating new projects

For Together and JBuilder to work together, you need both a Together (`.tpr`) and JBuilder (`.jpx`) for your project. For information on creating Together projects from within JBuilder, refer to the *Getting Started* guide located under **Help | Manuals | Getting Started** from Together' s main menu.

Using the New Project dialog

When using the New Project dialog to create a project, you have more control over the process.

Important If you are using Together Edition for JBuilder and JBuilder, you must refer to *Getting Started with* Together Edition for JBuilder for information on setting up a new project and working with existing projects. To view the manual, choose **Help | Manuals | Getting Started** from Together' s main menu.

To create a project using the New Project dialog, follow these steps:

1. From the File menu, choose **New Project**.

   Alternatively, choose **File | New** to open the Object Gallery, select the **New Project** icon, and click **Next.**

2. Complete the information required by the New Project dialog, as listed in Table 10.

Table 10 Preliminary information required by the New Project dialog

Field name	Description
Project name	Name of the project
Location	Location for Together to create the project file and initial and default diagrams; The Location field displays a default path. By default, all items in and under the project directory will be reverse engineered, and can be modified (unless specified read-only at the OS level).
Initial diagram	Type of diagram that Together will generate along with the new project; Select the type of diagram or None.
Show package dependency checkbox	Allows you to automatically draw dependency links between packages. Note that this slows performance for large projects. Rescan the project at another time using the Update Package Dependencies command on the diagram right-click menu.

3. Click **Next** to proceed.

   The project now contains one root directory — the primary root as specified in the previous step. At this point, you can click **Finish** to create the project if you do not need to include any other directories as part of the project. Or, proceed to the next step.

4. Use the next page of the New Project dialog to further control how Together sets up the project, including:

- Adding resources as detailed on page 79
- Removing resources, as detailed on page 81
- Setting resource options, as detailed on page 81

5. Click **Finish**.

After clicking Finish, Together creates the project file with the filename specified and the initial diagram (with the same name) in the specified directory (called the project directory). The project file has the `.tpr` extension and appears in the Explorer pane. At a later time, modify the project from the Project Properties dialog box as described in "Editing project properties" on page 76.

Editing project properties

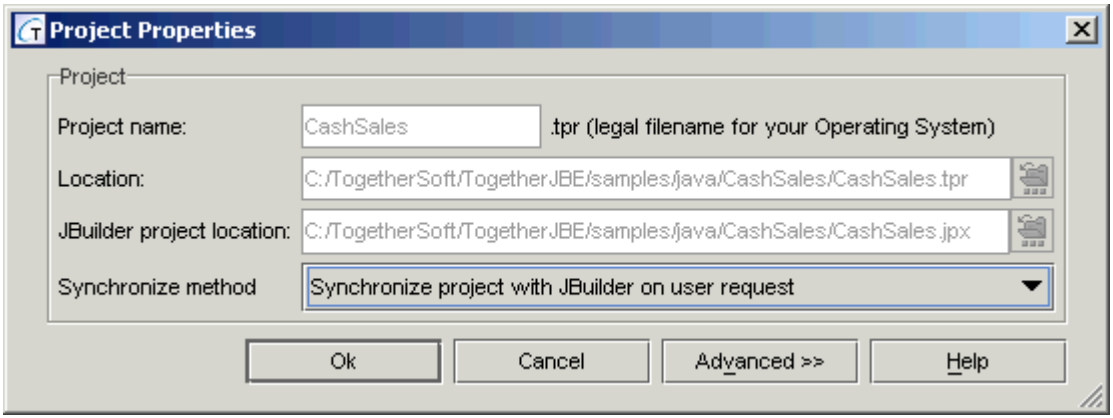
A project's properties include its name, location, and resources. After initially creating a project, view and edit its properties using the Project Properties dialog.

To access the Project Properties dialog, follow these steps:

1. Open a project
2. In the Explorer pane, choose the Model tab.
3. Right-click on the project name and select **Project Properties**. The Project Properties dialog appears in default mode, as shown in Figure 14.

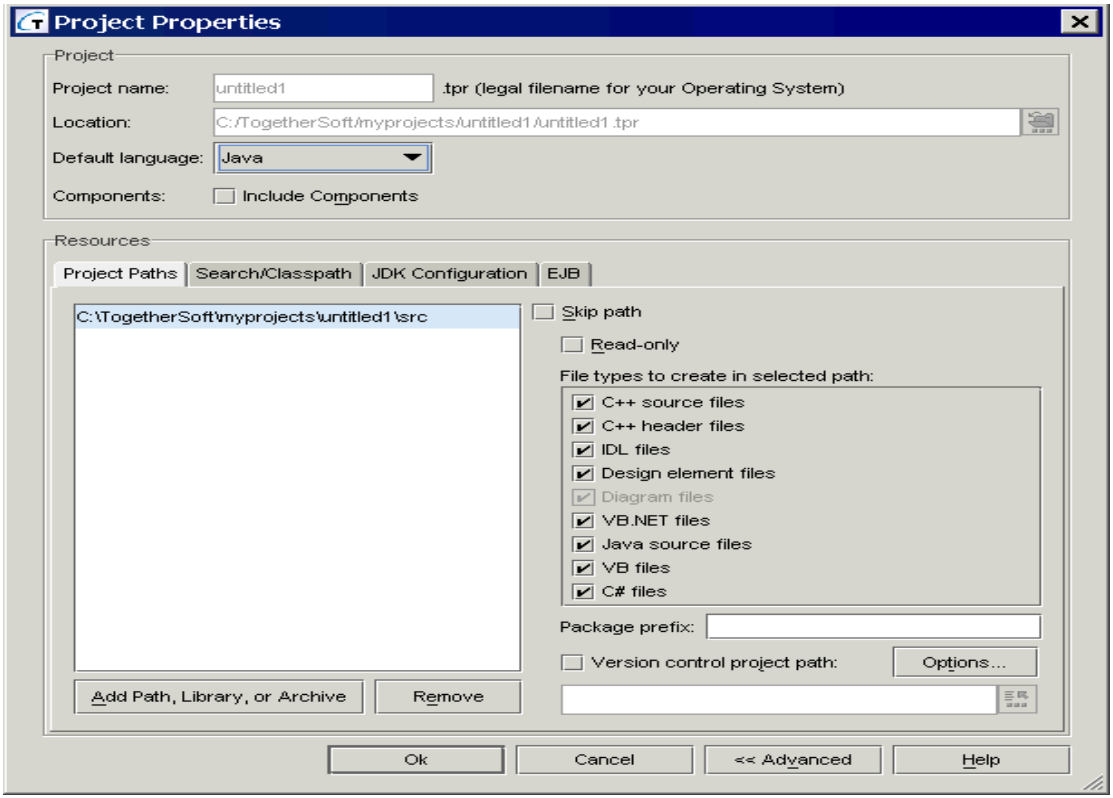
Note With the exception of the JBuilder *project location* and *Synchronize method* fields, Project Properties shows the same fields as the New Project dialog. The JBuilder-specific fields display in the Project Properties dialog once you have synchronized your Together and JBuilder project files. JBuilder-specific fields do not display in the New Project dialog. For a description of the fields found in the New Project dialog, see Table 10, "Preliminary information required by the New Project dialog" on page 75.

Figure 14 Project Properties dialog in default mode



4. For information about the JBuilder-specific property, *Synchronize method*, see "JBuilder-specific options" on page 78.
5. If necessary, access advanced mode by clicking **Advanced >>**. The Project Properties dialog appears in advanced mode as shown in Figure 15. For information about the properties that you can set in advanced mode, see the following sections:
   - "Adding Resources" on page 79
   - "Removing Resources" on page 81
   - "Setting Resource Options" on page 81
6. Edit properties as needed and then click **OK.**

Figure 15 Project Properties dialog in advanced mode.



JBuilder-specific options

This release of Together Edition for JBuilder enables you to synchronize projects from JBuilder to Together and vice versa. The JBuilder-specific fields, JBuilder *project location* and *Synchronize method*, display in the Project Properties dialog once you have synchronized your Together and JBuilder project files.

Synchronize method

An unsynchronized state is determined whenever there are changes in the JBuilder project properties. Changes to project source files are recognized automatically.

By default, Together notifies you when its project properties are not in sync with the corresponding JBuilder project properties. Specifically, a dialog prompts you to either keep both projects unsynchronized, or synchronize the Together project with the JBuilder project. The Confirm Synchronization dialog displays each time

the unsynchronized state is determined. However, you can disable this option at the project level by choosing **Project | Project Properties** and then setting *Synchronize method* to one of the following options:

- **Synchronize project with** JBuilder **automatically** - Project changes in JBuilder are automatically synchronized when opening a project in Together or switching the focus to Together, without asking for confirmation.
- **Synchronize project with** JBuilder **on user request** - (Default setting) When opening a project in Together or switching the focus to Together after making changes in JBuilder, a confirmation dialog asks whether you want to synchronize changes to the project. In each instance, you may choose to synchronize or not.
- **Never synchronize projects in future** - Together does not automatically synchronize project properties. You are not prompted to synchronize changes.

JBuilder project location

The JBuilder *project location* is a default setting created whenever Together and JBuilder projects are synchronized.

Adding Resources

The information in this section assumes that you have *either:*

- Accessed the Project Properties dialog in advanced mode, as described in "Editing project properties" on page 76; *or*
- Accessed the New Project Dialog, as described in "Using the New Project dialog" on page 75

The Project Paths and Search/Classpath tabs shown in Figure 15 display the lists of the directories and/or archive files currently included as resources available to the project. By default, the path specified in the Location field is present on the Project Paths tab, and the paths to the standard libraries are present in the Search/Classpath list. On the Search/Classpath tab, checking Include Classpath adds your classpath directories to the list.

For Java projects, the JDK Configuration tab contains information about the current JDK configuration. Choose the preliminary prepared JDK configuration from the drop-dawn list while creating a new project. To figure out how to create a new JDK configuration, see "Creating a new JDK configuration" on page 82.

For J2EE projects, choose the current application server using the EJB tab.

Items in the Project Paths list are considered as project roots. Roots contain compiled or source classes and/or Together diagrams. Project roots are parsed during reverse engineering and treated as modifiable unless you specify otherwise in the other controls. See "Setting Resource Options" on page 81 for more information.

The Search/Classpath tab displays resources that reside on the classpath (as defined in your environment) and on any other paths you want Together to search for resources. These resources are available to show in diagrams but are not part of the project. They are not parsed, and do not show as project content in the Explorer unless they are added to a diagram as a shortcut. Also, their content is not modifiable within the project context.

To add a resource directory to the project, follow these steps:

1. Click **Add Path, Library, or Archive** to display the *Select path, library, or archive* dialog box.
2. Navigate to the directory you want to include in the project.
3. Click **Open** or **Select**, depending on the OS you are running on.

Note that all subdirectories of the specified resource directory are included by default. You can exclude some specific subdirectories in one of the following ways.

- If you want the parsing engine to skip them but still want the project tracked by Together, add the individual subdirectories with the Add Path or Archive button, as described above and then check Skip path on the Project Paths tab (see Skip path).
- To exclude some subdirectories completely, use the Ignore files and folders option in the General options group of the Project Options dialog to specify the ignored directories.
- Sometimes the order of resources listed in the Search/Classpath is very important. You can move the members of the list up and down using the arrow buttons right.

In Java projects, some project resources may reside in compressed Zip or JAR archive files. Any new Java project automatically gets `rt.jar` as a "hidden" project root. The path to this archive is listed in the JDK Configuration tab along with JDK configuration libraries.

Note If you attempt to add `rt.jar` manually, a message saying "root added implicitly" displays. If you open an older Java project without `rt.jar` specified as a project root, you get a message and the classpath is searched for `rt.jar`, which is then implicitly added.

To add an archive file as a project resource:

1. Select either the Project Paths or Search/Classpath tab under Resources in the New Project (or Project Properties) dialog box.
2. Click the Add Path or Archive button to display the Select Path dialog.
3. Navigate to the directory containing the archive file and select a `.zip` or `.jar` file.
4. Click Open or Select, depending on the OS you are running on.

To reorder libraries and paths in the Search/Classpath tab, use the arrow buttons.

Removing Resources

The information in this section assumes that you have *either:*

- Accessed the Project Properties dialog in advanced mode, as described in "Editing project properties" on page 76; *or*
- Accessed the New Project Dialog, as described in "Using the New Project dialog" on page 75

Remove any resource from the Project Paths list or the Search/Classpath list by selecting the resource and clicking the Remove button. This is useful, for example, if you included your full Java classpath but it contains some directories that you really don't need in the new project. Removed resources are not deleted from disk, they are just not available to the project. If you need them again, add them later using the instructions in "Editing project properties" on page 76.

Setting Resource Options

The information in this section assumes that you have *either:*

- Accessed the Project Properties dialog in advanced mode, as described in "Editing project properties" on page 76; *or*
- Accessed the New Project Dialog, as described in "Using the New Project dialog" on page 75

After adding resources to the project, set options that control how they are parsed and how you access them when modeling. By default, parsing of all the paths in the Project Paths list includes all recognized file types, and all resources are parsed as part of the project and display in the Explorer's Model tab. To change the default setting, modify the treatment of each of the resources in the Project Paths list.

To set options for a resource, select it in the Project Paths list and modify any or all of the following:

- **Skip path:** Selecting this option causes the parsing engine to skip the selected resource, but the resource is still tracked with the project by Together and may be included in documentation generation or accessed as part of model information by modules.
- **Read only:** Checking this box makes the contents of the resource a read-only part of the project. It displays as project content in the Explorer's Model tab but is not modifiable from within the project. When checked, File types and Package prefix options are disabled.
- **File types to create in selected path:** Check only those file types you want the parsing engine to work on. This can speed parsing of resources having different types of content. Use the default (all file types) unless you want to skip source files, or you want to specifically exclude compiled classes.

- **Package prefix:** Specify the exact name you want Together to use for package statements referencing the selected resource.

Creating a new JDK configuration

For Java projects, you must invoke some JDK x.x. You can create different JDK configurations and then use them if needed.

To create a new JDK configuration:

1. From the **Tools** menu, choose **Configure JDK**. The *JDK Configurations Editor* dialog opens.

Note *Configure JDK* is an activated feature. If you don't see *Configure JDK* in the list of *Tools* options, choose **Tools | Activate/Deactivate Features**, and check JDK Configuration in the *Activate/Deactivate Features* dialog.

2. Type the full path to the `%JAVA_HOME%` in the *Directory to start search* field, check *Perform search recursive* if necessary, and click **Search**.

   For filling the *Directory to start search* field, you can alternatively use the file chooser.

3. If `java.exe` is successfully found out using the new path, the corresponding line appears in the list of the available JDK configurations.

4. To remove the useless JDK configuration from the list, click **Remove**.

To choose the JDK configuration as a default one:

1. Highlight the chosen line in the list of the available configurations and click **Set Default.** The default JDK home and the default JDK version are displayed. The corresponding `rt.jar` file appears in the list of the default JDK libraries.

2. Use the *Add/Remove* buttons to add/remove the additional libraries if needed.

Note The default value of the JDK configuration after installing Together is the bundled JDK located in `%TG_HOME%/jdk`.

To edit a JDK configuration, you can also use the *Edit JDK Configuration* button in the Project Properties dialog.

Running multiple project instances

By running multiple instances of Together, you can work with several projects (or files associated with Together). Note that you can run multiple instances Together, but you can run only one project in each instance.

Configure options to control how Together reuses instances. For example, select one of the following options:

- **Reuse current instance**: File always opens in the current instance of Together (without requesting confirmation). If this requires closing the project, the project is closed without any warning.
- **Run new instance**: File always opens in a new instance of Together (without requesting confirmation).
- **Always confirm reuse**: Before opening a file, a dialog opens that asks whether you want to reuse the current instance or start a new instance of Together.
- **Confirm reuse only if project must be closed**: Together asks for confirmation only when reusing the current instance requires closing the current project. If the file can be opened without closing the current project, it is automatically opened in the current instance of Together.

To set options for multiple project instances, follow these steps:

1. From the Tools menu, choose **Options | Default Level**.
2. From the tree of the Default Options dialog, select **General**.
3. Select a value for *Reusing* Together *instances*.
4. Click **OK** and then **Apply**.

Note This option setting overrides the option *Confirm when closing project*.

Setting up large projects

A project can contain large code bases comprised of dozens of directories and hundreds of classes. Such large projects often consist of subsystems, with modules or components within the subsystems. Together translates this to projects and subprojects.

Instead of creating a single project that encompasses an entire code base, identify the subsystems and the modularity within them, creating a number of Together projects in key directories of particular interest or significance. As an example, refer to the following project located under your Together installation: `$TGH$/modules/components/CoadModelingComponents/all.tpr`. Although this is not a particularly large code base, it illustrates the project-subprojects organization you can use for larger projects.

Also, consider automating documentation generation for very large projects. Write a script or module to regenerate all documentation or a set of scripts to update different parts, which run automatically at different times. See Chapter 11, "Generating Documentation for Together Projects" for more information.

Creating views with referenced content

Together's project definition feature allows you to create projects with content that is purely logical, containing only the items you need to see for a particular purpose. For example, you could create a directory that primarily provides a view of your code base. Under such a directory, you can create a series of project folders that contain Together projects that bring in different parts of your code (for example, only abstract classes in your problem domain).

When creating such projects, specify the subdirectories to use as resource roots. Other directories with classes (and other elements) that you might want to show (but not modify) can be specified in the Source/Classpath list. See "Creating new projects" on page 75 for more information.

Performance tuning

When working on a project with a large code base, consider the following issues that may impact performance:

- Check the resource paths used by the project. Deep path names such as `C:` can hinder performance. For instructions on how to view resource paths, see "Editing project properties" on page 76.
- Check the memory used while parsing. If memory increases close to the maximum, try to increase the swap file size. (Set the minimum swap file size in the range of 100-150M).
- Try running Together with `jre.exe`, the Sun virtual machine (VM) that provides options for specifying the Java heap size. Change the maximum heap size (`-mx32m`) to a larger amount (such as `-mx60m` or `-mx100m`, depending on the availability of virtual memory). Make sure your computer hardware has sufficient power and system resources to handle the values you specify in order to achieve the effect you want.
- Monitor and manage the memory usage via the *Memory Status* dialog. To open this dialog, use the keyboard shortcut Ctrl+Shift+T. Use the *Garbage Collect* button to release the unused memory.

Chapter

5

Reverse Engineering

A central feature of Together is LiveSource — the ability to immediately synchronize class diagrams with the implementation code.

LiveSource means that your UML class diagrams are always synchronized to the source code that implements them. When you change a class diagram, Together immediately updates the corresponding source code. When you change the code, Together updates the visual model. There is no intermediary repository, no batch code generation, and no risk of losing code.

The LiveSource feature applies to existing code as well as code that is being developed. Together can *reverse engineer* source code, building a model around existing code or restoring a model from archived files.

This chapter includes the following topics:

- "Reverse engineering techniques" on page 85.
- "Recognizing links during reverse engineering" on page 87.
- "Reverse engineering archive files" on page 88.

Reverse engineering techniques

You can create a new project using existing code or incorporate the code into an existing project. The existing code can be source code, *.jar files, or *.zip files.

Creating a new project with existing code

To create a new project with existing code, use the New Project dialog. You can specify the use of existing source files or JAR/ZIP archives.

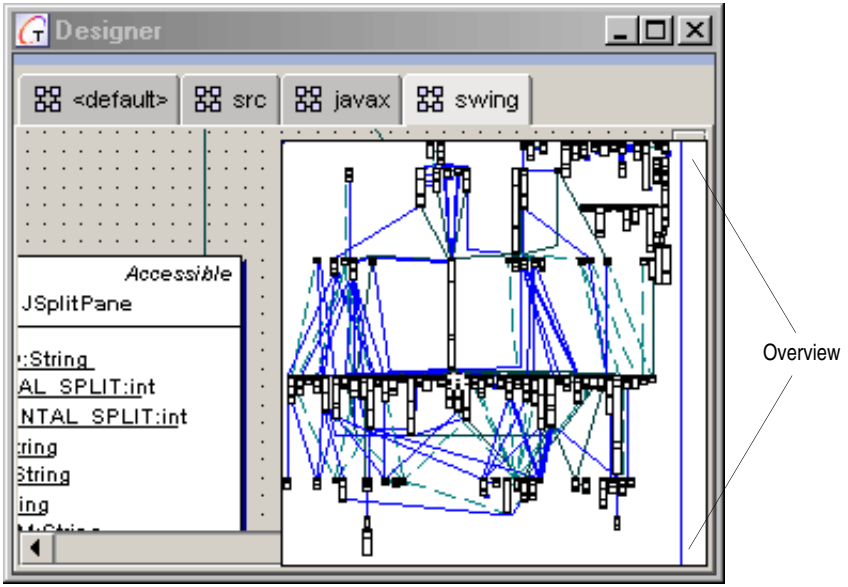
To reverse engineer code with the New Project dialog:

1. Choose **File | New Project** from the main menu. The *New Project* dialog opens.
2. In the New Project dialog, enter the project name and location. For the initial diagram type, select **Class**.
3. Click **Advanced**.
4. Choose the **Project Paths** tab.
5. Click the **Add Path or Archive** button and then choose the file that contain the source code that you want to use. Repeat this step to choose additional files if necessary.
6. Click **Finish**. Together uses the code of the specified files to create the new project.

Together generates a set of class diagrams according to the structure of the source code (physical class diagrams). The `<default>` diagram corresponds to the top level package of the project.

Figure 16 shows the results from reverse engineering JDK 1.3. The Designer pane displays the swing library. The overview, as shown, indicates the extent of the library and the connections among its classes.

Figure 16 Overview of class diagram for Swing



Note You can create a new project with existing source code without using the *New Project* dialog. In that case, however, the physical class diagram files are stored in the same folders as the source files.

Together can reverse engineer code that you can incorporate into an existing project. The code can be source code, `*.jar` files, or `*.zip` files.

To reverse engineer code by incorporating it into an existing project:

1. Open the project.
2. Select **Project | Project Properties** from the main menu.
3. In the Project Properties dialog, choose the **Project Paths** tab.
4. Click the **Add Path or Archive** button and then choose the file that contain the source code that you want to use. Repeat this step to choose additional files if necessary.
5. Determine if Together should treat the code as read-only (default):
   - Together treats *.jar and *.zip files as read-only.
   - If you are incorporating source code into the project, un-check **Read Only** if you want to be able to modify the source code or add files to the source code folders. You can check the types of files to create in the project path.

Note If you un-check **Diagram files**, Together creates new folders inside its project directory to hold the diagram files that it generates.

6. Click **OK**. Together incorporates the codes into the specified project.

Use the Model tab of the Explorer to open the diagrams for the reverse-engineered code.

Tip When you reverse engineer Java code, run the audit *Detect Collection-Based Associations* (under *Documentation* audits) so that Together can display associations for collections without your manually inserting `@association` tags into the code. This auto-fix audit inserts the tags for you.

Alternatively, to add existing Java code to an existing project:

1. Copy the necessary Java files to the project source path directory. If you are copying EJBs, copy Java code of their implementation classes, home and remote interfaces.
2. Correct the names of packages in the incorporated Java code.
3. Reopen your project. The design elements corresponding to the incorporated code appear on the diagrams according the packaging. *Generalization* and *Association* links between elements appear, as well.

Recognizing links during reverse engineering

Code generated from round-trip engineering is defined by codegen options and patterns (including textual templates). Such options and patterns define the creation of the implementation code of UML diagrams.

Reverse engineering implies that the source code is analyzed by the parser. As such, encountered classes are passed to the model as is. In contrast, links are created by the following source code elements:

- **Attributes.** A special engine parses the attributes of a class, selects certain tokens from the attribute string, and chooses the appropriate blueprint allowing the recognition of a link with its properties. In particular, the link type and destination class are recognized and the attribute modifiers are ignored.
- **Tags in doc comments.** Define links and optionally, their properties.
- **Links as comments.** Not visible to the compiler, as they are commented.

The configuration system of Together contains these *parser blueprints*. They are used for the automatic recognition of links in classes during reverse engineering and actually determine which elements of the source code produce links.

The blueprints are defined in the configuration files for the supported languages as follows:

```
$TGH$/config/<language>.config
```

```
The format of a blueprints is:
```

```
parser.<language>.blueprint.link.<link type> = <link expression>
```

```
Each attribute declaration or design comment is compared against the
blueprint pattern and if recognized, forms a link of the corresponding
type.
```

The first matched blueprint is used. Optionally, the blueprints can include additional sub-properties (tags), in the following format:

```
parser.<language>.blueprint.link.<link type>.<link property> = <property
value>
```

The <link type> is not actually used, but recommended to make the description of parser blueprints meaningful.

Note Tag values, directly specified in the Inspector, override those from the blueprint.

You can define your own blueprints with their sub-properties, specifically designed for certain applications. However, the parser engine scans the entire set of existing blueprints. Thus, too large a number of custom blueprints can impact performance.

Reverse engineering archive files

Together extends reverse engineering to J2EE archive files. It is possible to restore the diagram structure of an existing *.jar, *.ear or *.war file. If you have implemented and deployed EJB, Enterprise Application, or Web Application modules, you can include them to your Together project. In process of reverse engineering, Together generates the corresponding diagrams with the relevant visual design elements.

To reverse engineer a diagram from an archive file:

1. On the main menu choose **File | Import | J2EE Archive**. The J2EE Module Import dialog appears.

2. For *J2EE module location*, specify the fully-qualified name of the source archive file to be imported. The archive file extension should be .jar, .war, or .ear.
3. For *Extracted module location*, enter the fully-qualified path to the target folder where the new diagram will be created. Note that the destination folder should not be the parent directory of the project source path.
4. Click **Ok**. Together restores appropriate diagram.

Note The Ok button is disabled unless archive type and destination directory are properly specified. Use file chooser buttons to define the required locations in the file system.

The extracted module corresponds to the archive file type. Thus, Together restores the structure of Web Application and EJB Assembler diagrams included in an Enterprise Application.

Tip If you are importing a module with JSP files, mark the files as JSP in the deployment descriptor so that Together restores them as JSP files. Otherwise, Together treats them as ordinary web files.

Chapter

6

Importing and Exporting Information

Together reads and generates information in multiple formats. This chapter explains many of the import and export operations that Together supports, focusing on DDL, IDL, and XMI.

This chapter contains the following topics:

- "Overview of import and export operations" on page 91.
- "Importing and exporting JDBC database information" on page 92.
- "IDL Export and LiveSource" on page 95.
- "Importing and exporting XMI models" on page 96.
- "Exporting a Together project to a JBuilder project" on page 98.

Overview of import and export operations

Together supports several different kinds of import and export operations:

- Import/export model information to XMI.
- Import/export to JDBC databases via DDL files.
- Generate Interface Definition Language (IDL) for a project.
- Export a Together project to a JBuilder project.

Importing and exporting JDBC database information

Information can be imported from JDBC-enabled databases into Together. Additionally, users can export model information from Together to create model-based database schemas. Together supports the following RDBMS types:

- mySQL 3.23
- Sybase AS Anywhere 6.x / 7.x
- IBM DB2 6.1 / 7.x
- SequeLink/Oracle
- InterBase 7
- JDataStore 6 Server
- Sybase AS Enterprise 12
- Oracle 8.x/.9
- Access 97/2000 (ODBC)
- MS SQL Server 7.0 (ODBC)
- Cloudscape 3.5

Activating database import and export

DB Modeling is an activatable feature for importing from a database or exporting to DDL.

To activate DB Modeling:

1. Open a project.
2. Choose **Tools | Activate/Deactivate Features** on the main menu.
3. In the resulting dialog box, go to the Together **Features** tab.
4. Check the box in the Activate column to the left of the module named **DB Modeling**. Click **OK** when finished.

Tip Deactivate any features you do not need and activate any others you do need simultaneously. Deactivate unused features in order to free up system resources.

With DB Modeling activated, the main menu contains two database import/export commands: **Tools | Generate | DDL** and **File | Import | Database schema**.

Generating DDL (exporting to DDL)

With Together's DB Modeling feature, you can generate a data definition language (DDL) file from a class diagram or entity EJBs. A compliant DBMS system can read the DDL to create a table schema. Together gives a choice of generating DDL scripts only or generating the DDL scripts and then running them in the same operation.

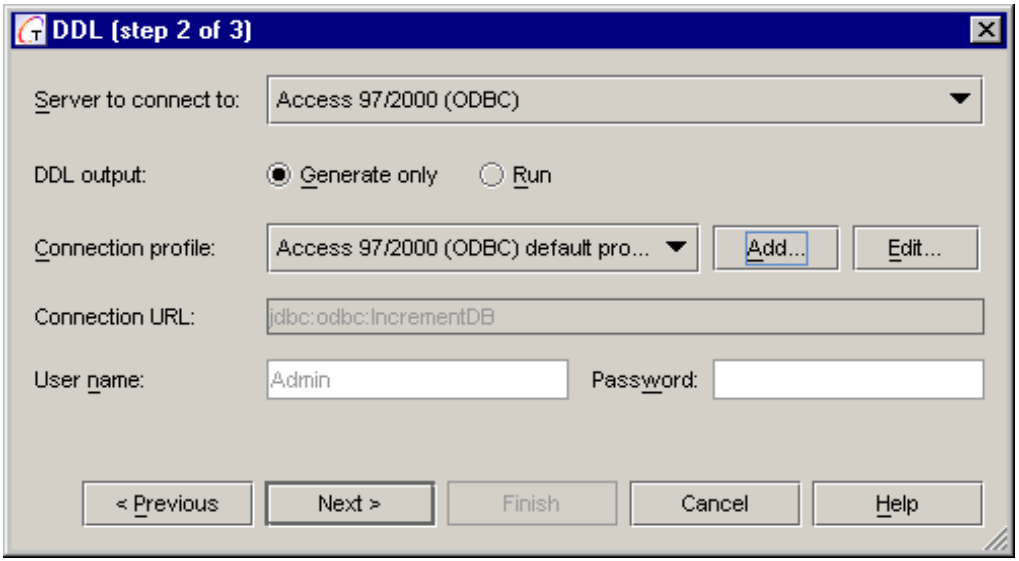
To generate DDL:

1. Open a project containing the diagram with the source for information to export.
2. Open the class diagram or the class diagram with an entity EJB that contains the information for the database schema.
3. On the main menu, choose **Tools | Generate | DDL**. (The DB Modeling feature must be activated for this command to be on the menu.)
4. The resulting DDL dialog box has 3 panes for specifying the details of the DDL generation.
    1. **Source model /code information**: Indicate class diagram or entity EJBs and the corresponding diagram for the source of information.
    2. **Target database**: Enter the specific database information. The DDL script generated corresponds to that database schema. Figure 17 shows the second pane after creating a proper connection profile.

        If you select **Run** from **DDL output**, you must select a **Connection profile** to an actual database that Together can access. You can click **Add** to create a new profile or edit an existing one.
    3. **DDL file**: Enter the name and location of the generated DDL file. If desired, edit the script. Click **Finish** to generate the DDL script.

        If you select **Run** from the **DDL output**, File name field is not available. In some cases (for example, for MS SQL Server), you can choose Schema name.

Figure 17 Exporting to DDL



Tip If you are exporting DDL from a class diagram, all classes in the diagram must be "persistent." Set the persistence for each class through its Properties (accessed via the right-click menu.)

Important MySQL does not support non-English characters.

Importing database meta-information

The DB Modeling feature enables you to import schematic information from a supported database to create a class diagram or entity EJBs.

If the database does not already exist, you should use your DBMS administration utility to create a JDBC/ODBC data source of the source database prior to importing.

To import a database:

1. Open a Together project or create a new project in which to store the imported information.
2. Choose **File | Import | Database Schema** from the main menu. The DB Modeling feature must be activated for this command to be on the menu.
3. The resulting Database Schema dialog box has 3 panes for specifying the import details.
    1. **Source database**: Choose the supported DBMS, set a connection profile, and connect to the database.

2. **Tables**: Select each table that you want to import from the left frame (Available Tables) and add it to the right frame (Tables for Import).
3. **Import to**: Select the kind of import (EJBs or class diagrams), and the diagram to store the information.Use an existing diagram or create a new one.

4. Click **Finish** to complete the import.

IDL Export and LiveSource

If a Together project contains class diagrams appropriately structured to support IDL, you can generate IDL for specific diagrams, classes, packages, or for the entire Together project. LiveSource enables you to generate IDL synchronized with your diagrams.

IDL Export Support is an activatable Together feature that is deactivated by default. You can activate it in the usual way, as described in "Activating database import and export" on page 92.

When the IDL Export Support is activated, the main menu contains the command **Tools | Generate | IDL**.

Exporting IDL from class diagrams

After activating IDL Export Support, generate IDL files from your projects.

To generate IDL:

1. Open the project containing the source to be exported to IDL.
2. If you want to generate IDL for a specific package, open its diagram in the Designer pane. (This is not necessary for generating IDL for the entire project.)
3. Choose **Tools | Generate | IDL** from the main menu.
4. In the resulting dialog box, select:
   - The package structure: **Current** (package), **Current with subpackages**, or **All** (the entire project).
   - IDL type: **COM IDL** or **CORBA IDL**.
   - Target directory: for the generated files.
5. Optionally review and change the IDL Options offered in the dialog. Click **Options** and use the on-screen help to change any option settings. Click **OK** to make any changes and close the IDL Options dialog box.
6. Click **OK** in the Generate IDL dialog box to generate the IDL.

Tip Open the message pane before generating IDL to view the messages generated by the process.

LiveSource support for IDL

Together supports LiveSource IDL engineering. You do not need to import IDL into a project. Simply create a new project specifying the location of your existing IDL code as one of the project's resource roots and continue working with it in Together.

To create a model on the base of the existing IDL source code:

1. On the main menu, choose **File | New | New Project**.
2. In the **Project Path** tab, click the button **Add Path, Library or Archive**.
3. Select the root directory that contains the source IDL code.
4. Click OK to start creating the new project around the specified IDL source.

Result: the newly created project contains the model, generated on the base of the specified IDL.

Importing and exporting XMI models

Import a model described by an XMI file into a Together project. This generates source code in the primary programming language of your Together product. You can export model information from a Together project to an XML file as well.

Note XMI import-Export requires Sun JRE version 1.2x or above

XMI import and export require activating the **Import-Export** feature. (Activating a feature is described in "Activating database import and export" on page 92.) When Import-Export is activated, the main File menu has Import and Export commands.

Importing XMI models

If you are importing a large model, close other applications to free up memory.

To import a model described in an XMI file:

1. Create a Together project for the model or open an existing project to which you want to add the XMI model. Create or navigate to the desired package.
2. On the main menu, choose **File | Import | Model from XMI File**.
3. In the resulting dialog box, select the XMI file and click **OK**.

The amount of time you must wait while Together processes the XMI code depends on the size of the model.

When Together finishes processing, it creates a new package diagram and opens it in the Designer pane. The Model tab of the Explorer displays the packages and the default class diagram for each package. When opening diagrams and selecting classes, use the JBuilder Editor to display the generated source code.

Together creates a log file with information about import operations and places it in the source folder of the imported xml file.

Note If you plan to export a model from Rational Rose® to XMI and later import it to Together, you should choose the ASCII/MBCS option in the Character Set options in Rose's Unisys XML Export dialog. Models exported to other character sets will not be properly imported into Together.

Exporting XMI models

Export a Together model to an XMI file. Together supports UML 1.1 and UML 1.3 Unisys XMI interchange for all standard UML diagrams, IBM XMI Toolkit, and OMG XMI.

To export a Together model to XMI:

1. Open the Together project you want to export.
2. On the main menu, choose **File | Export | Model to XMI File**.
3. In the resulting dialog box:
   - Select the target location and specify the filename for the generated XMI file.
   - Select encoding of the output stream from the drop-down list.
   - Select the XMI type from those listed, then choose the option specifying which diagrams from the current project you want exported to XMI.
4. Click **OK** to start the export process.

The time required to generate the XMI code depends on the size of the model. Together displays the progress of the operation on its status bar.

Figure 18 shows the Export to XMI dialog box, which gives eight options for the XMI type.

When exporting a model to Rational Rose, keep in mind that some of the elements supported by Together are not allowed in Rose. These are, for example, objects on the Class, Activity, State, Component or Deployment diagrams, or components and interfaces on the Deployment diagram. The second option of the Export to XMI dialog (XMI 1.1 for UML 1.3 Unisys Extension, Recommended for Rose) provides the ability to drop all Rose-inconsistent elements. If such elements are encountered, appropriate messages appear in the Message pane. For example: "The following element is not exported due to Rational Rose limitations: Object aSale in ClassDiagram Object View"

User configurable translations

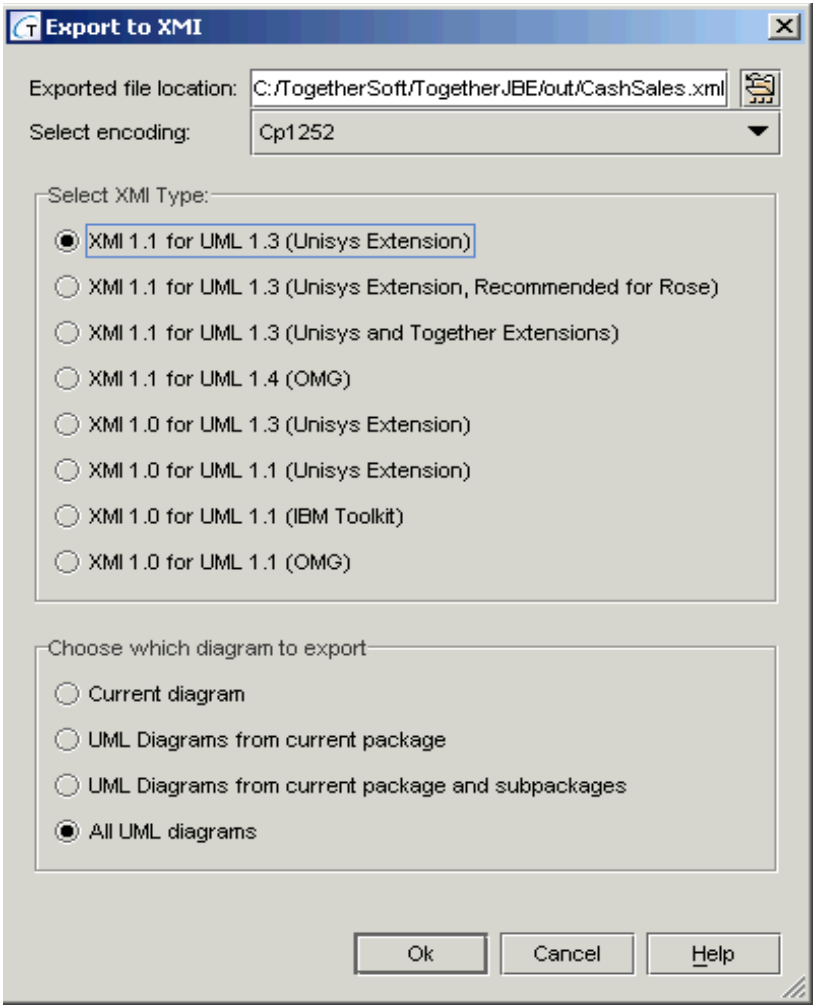
To support compatibility between Together model tags and XMI tags, the interchange module provides a list of translations that contain entries in the format:

```
Tag_Name = RationalRose$MDC/:Tag Name
```

To import or export some specific properties of your project from or to XMI, edit this config file:

```
%TGH%/modules/com/togethersoft/modules/interchange/InterchangeTags.config
```

Figure 18 Export to XMI dialog box


Export to XMI
Exported file location:
C:/TogetherSoft/TogetherJBE/out/CashSales.xml
Select encoding:
Cp1252
Select XMI Type:
XMI 1.1 for UML 1.3 (Unisys Extension)
XMI 1.1 for UML 1.3 (Unisys Extension, Recommended for Rose)
XMI 1.1 for UML 1.3 (Unisys and Together Extensions)
XMI 1.1 for UML 1.4 (OMG)
XMI 1.0 for UML 1.3 (Unisys Extension)
XMI 1.0 for UML 1.1 (Unisys Extension)
XMI 1.0 for UML 1.1 (IBM Toolkit)
XMI 1.0 for UML 1.1 (OMG)
Choose which diagram to export
Current diagram
UML Diagrams from current package
UML Diagrams from current package and subpackages
All UML diagrams
Ok
Cancel
Help

Exporting a Together project to a JBuilder project

Export a Together project to JBuilder project. You may find this useful in the event that you begin modeling in Together before creating a JBuilder project.

To export a Together to a JBuilder project:

1. Open the Together project you want to export.
2. On the main menu, choose **File | Export | Project to JBuilder Project**.
3. The resulting dialog box automatically selects:
   - The Together project.
   - The JBuilder project name.
   - The JBuilder project home directory.
4. Accept the defaults, and click **OK** to export the project.

Alternatively, you can select the Together project in the Model tab of the Explorer. Right click on the project and choose to **Open Project in** JBuilder. If a JBuilder project does not exist, then you can choose to create a project, and synchronize it with the current Together project.

Part

3

MODELING WITH TOGETHER

- Chapter 7, “Introduction to Modeling”
- Chapter 8, “Working with Diagrams”
- Chapter 9, “UML Modeling”
- Chapter 10, “UML Extension Diagrams”

Chapter

7

Introduction to Modeling

The primary objective of modeling is to organize and visualize the structure and components of software intensive systems. Models visually represent requirements, subsystems, logical and physical elements, and structural and behavioral patterns.

While contemporary software practices stress the importance of developing models, Together extends the benefits inherent to modeling by fully synchronizing diagrams and source code.

This chapter explains how the topics of modeling and diagrams are organized in this *User Guide*. It includes the following topics:

- "Overview of modeling" on page 103.
- "Diagrams supported in Together" on page 104.

Overview of modeling

In the *UML User Guide*, Booch *et al* cite a number of activities involved in modeling a system's architecture. These activities include:

- Identifying different architectural viewpoints such as use case, design, process, implementation, and deployment views
- Identifying the system context and the actors involved
- Decomposing a large, complex system into more granular subsystems.

In addition, the authors outline various views that apply to both the overall system and each of the subsystems:

Use Case view: Represents use cases describing system behavior as seen by analysts, end users, and testers. Models consist of Use Case diagrams for the static aspects of a system and appropriate combinations of Activity, Collaboration, Sequence, and State diagrams for dynamic aspects.

Design view: Represents a design view specifying classes, interfaces, and collaborations. These provide a working vocabulary for the system in terms of problem and solution. Models consist of Class diagrams (including Objects as necessary) to represent static aspects of a system with appropriate combinations of Activity, Collaboration, Sequence, and State diagrams for dynamic aspects.

Process view: Represents a process view to describe the threads and processes of synchronization and concurrency mechanisms. Models consist of diagrams used for the Design view with active classes and objects representing threads and processes.

Implementation view: Represents the components used to build and release the system. Models consist of Component diagrams for the static aspects of a system with appropriate combinations of Activity, Collaboration, Sequence, and State diagrams for dynamic aspects.

Deployment view: Represents the nodes, components, and interfaces forming the hardware topology for the runtime system. Models consist of Deployment diagrams for static aspects of a system with appropriate combinations of Activity, Collaboration, Sequence, and State diagrams to for dynamic aspects.

Patterns: Represent the architectural and design patterns of each of the previous models with appropriate diagrams to show collaborations.

Diagrams supported in Together

Together supports all of the Unified Modeling Language (UML) diagrams in addition to specialized diagrams for modeling business data, business processes, and robustness analysis. Table 11 lists the diagrams supported in Together and indicates where to find respective information in this guide.

Tip For information on working with specific diagrams, see the references in Table 11. For instructions that apply to all diagrams such as creating, editing, and hyperlinking diagrams see Chapter 8, "Working with Diagrams".

Table 11 Diagrams supported in Together

This section...	Includes these diagrams...

Table 11 Diagrams supported in Together (continued)

Chapter 9, “UML Modeling”	“Use case diagrams” on page 138
	“Class diagrams” on page 140
	“Sequence and collaboration diagrams” on page 147
	“Statechart diagrams” on page 156
	“Activity diagrams” on page 159
	“Component diagrams” on page 162
	“Deployment diagrams” on page 164
Chapter 10, “UML Extension Diagrams”	“Business process diagrams” on page 167
	“Robustness analysis diagrams” on page 168
Chapter 19, “J2EE Support”	“EJB assembler diagram” on page 302
	“Enterprise application diagram” on page 302
	“Taglib diagram” on page 303
	“Taglib diagram” on page 303
	“Resource adapter diagram” on page 303

Chapter

8

Working with Diagrams

This chapter outlines basic techniques for working with diagrams. It covers the following topics:

- "Creating diagrams in projects" on page 107
- "Drawing diagram elements" on page 110
- "Working with elements and links" on page 117
- "Viewing diagrams and managing diagram layout" on page 121
- "Editing diagrams" on page 123
- "Hyperlinking diagrams" on page 127
- "Annotating diagrams" on page 130
- "Standalone design elements" on page 131
- "Saving and copying diagram images" on page 132
- "Printing diagrams" on page 133

Creating diagrams in projects

Diagrams exist within the context of a project. You create or open a project before creating any new diagrams. If you create a new project around an existing code base, Together automatically generates class diagrams showing the contents of each package when it parses your code. Before creating a project, you may customize forward and reverse engineering and/or source code formatting (see Chapter 2, "Setting Your Personal Preferences" on page 49).

Together diagrams fall into two basic categories:

- UML diagrams (class, use case, sequence, and so on)

- Special Together diagrams (EJB assembler, business process, and others)

Some diagrams are source-generating. These are: class diagrams, sequence and collaboration diagrams, and some J2EE-related diagrams. The contents of such diagrams are synchronized with the source code. Right click a class or interface symbol on a diagram and choose Edit to open the source code in the JBuilder editor.Selecting a class symbol in the diagram opens the underlying class in the special tab of the Editor, marked with the class name. If the class is read-only, the tab is also marked with the lock icon. Selecting an element within a class or interface symbol automatically navigates to the appropriate line of the source code.

When you begin a new project or add new diagrams to an existing project, you can create diagrams using one of the following:

- Main menu or toolbar
- Hyperlinking feature
- Clone command

The created diagrams are stored in text files with the extension `*.dfDiagramType`. For example, the file `name.dfClass` corresponds to a class diagram, and `name.dfUseCase` corresponds to a use case diagram.

Using the main menu or toolbar to create diagrams

To create a new diagram in a project:

1. Select the destination directory or package on the Directory or Model tab in the Explorer.
2. Choose **File | New** on the main menu to open the Object Gallery.

Tip Alternatively, click the New button on the main toolbar.

3. In the *Templates* pane of the Object Gallery, choose the **Diagram** icon and click **Next** to open the New Diagram dialog.
4. Choose the **UML** or Together tab as required, and click the icon for the type of diagram you want to create.
5. In the *Diagram name* field, type a name for the diagram.
6. In the *Package name* field, specify the destination package for the new diagram file. The default destination is the package currently selected in the Explorer.
7. Check *Include in current diagram* if you want to add an element to the current diagram with a symbol representing a logical link to the new diagram.
8. Click **Finish**. The new diagram opens in the Designer pane.

Using the hyperlink feature to create diagrams

The hyperlink feature enables you to create a new diagram that automatically links to an existing diagram or diagram element.

To create a new hyperlinked diagram:

1. Open an existing diagram from which to create the hyperlink (or create a new diagram).
2. Select the element or group of elements to which you want to link the new diagram. If you want to link to the diagram as a whole, click on the diagram background.
3. On the Selection menu, choose **Hyperlink To | New Diagram.** The New Diagram dialog opens.
4. Specify the type and location for the new diagram as described above. Click **Ok**.

For more information, see "Hyperlinking diagrams" on page 127.

Cloning diagrams

The Clone command lets you create a new diagram with the same content as the existing one.

To clone a diagram:

1. On the Model tab of the Explorer, select the diagram to be cloned.
2. Choose **Selection | Clone** on the main menu.

Tip Alternatively, choose the Clone command from the right-click menu of an existing diagram node on the Model tab of the Explorer.

The new diagram is created with a unique default name in the same package as the original diagram.

Renaming diagrams

To rename a diagram using the Inspector:

1. Open the diagram for editing, and make sure the Inspector is open.
2. Click on the background to display diagram properties in the Inspector.
3. Modify the *Name* property in the diagram Inspector.

To rename a diagram in place:

1. On the Model tab of the Explorer, select the desired diagram node.
2. Double-click the diagram name or press the **F2** button to activate the in-place editor.
3. Change the name as required and press **Enter.**

Configuring diagram options

The Diagram Options dialog provides a number of customization settings. You can set these options either globally for all diagrams, project-wide for just the current project, or locally for the current diagram.

To configure the diagram settings:

1. Choose **Diagram Options** on the diagram right-click menu. The Options dialog opens on the Diagram level.

Tip Alternatively, choose **Tools | Options | Diagram level.**

2. Change the configuration level if necessary.
3. Specify diagram options, following the instructions in the Description pane of the Options dialog.

Note that the Diagram Options dialog provides synchronization options with JBuilder.

Options for improving graphics

To make your diagrams look better, note the options that help improve graphics.

3D look

To make your diagram look three-dimensional, open Diagram Options and check *3D look*. This draws a shadow under each diagram element to create a three-dimensional effect. This option applies only to those elements that have the *3D look* option set to *default* on the View tab in the Inspector.

Anti-aliasing

When zooming in a diagram, you might observe rough lines, particularly on links. To smooth the lines being drawn, open Diagram Options and check *Antialias graphics*.

Drawing diagram elements

This section covers the basic techniques for placing diagram elements and annotations into diagrams and drawing relationship links between them.

When you create a new diagram, the Designer pane presents an empty background. You place the various elements (such as class, use case, and so on) on the background and draw relationship links between them according to the requirements of your model.

The main tools for drawing and editing diagram elements are:

- **Diagram toolbar buttons.** Use toolbar buttons to place elements on the diagram and draw links to connect them.
- **Diagram right-click menu.** Right-click on the diagram background to show the diagram right-click menu. Use the right-click menu to show hidden objects, manage the layout, configure diagram-level options, and update diagrams and hyperlinks.

- **Element right-click menus.** The right-click menus of the various elements and links provide commands specific to each. For example, you can add or delete members (or delete the element itself), cut-copy-paste, hide and show elements, route links, and more. Explore the right-click menus of the different elements as you encounter them to see what is available for each one.
- **Inspectors.** (Right-click menus | Properties) Use Inspectors to edit diagram or element properties, and to create hyperlinks to diagrams, elements of diagrams and other diagrams, and to diagrams or elements and files or URLs. You can also edit annotations and source code comments.

Undo/Redo

All operations are reversible. Undo and Redo commands can affect the visual diagram, layout of elements, addition of attributes and operations, and so on. These features are *not* available when the change to the diagram involves creating or renaming a file or directory, such as creating and renaming classes, interfaces, or packages.

- To undo the last change you made to the diagram, choose **Edit | Undo** on the main menu, or press **Ctrl + Z** .
- To restore the last change you made using Undo, choose **Edit | Redo** on the main menu, or press **Ctrl + Y**.

Tip The size of the Undo/Redo buffer is configurable in the global or project-local `workspace.config` file.

Using the grid

You can optionally display or hide a design grid on the diagram background and have elements "snap" to the nearest grid coordinate when you place or move them. Grid display and snap are enabled by default.

To control grid parameters:

1. Choose **Tools | Options | <level >** on the main menu (where <level> is Default Level, Project Level, or Diagram Level, depending on how broadly you want the settings to apply).
2. Expand the *Diagram* node of the Options dialog and select the *Grid* node.
3. Set the *Show grid* and *Snap to grid* options as desired.
4. Specify the granularity of the grid in the fields *Grid width* and *Grid height*, in pixels.

Drawing elements

To place an element on a diagram:

1. On the diagram toolbar, click the button for the element you want to place in the diagram. (Icons are identified with tool tips.) The button stays down.
2. Click on the diagram background in the place where you want to create the new element. This creates the new element and activates the in-place editor for its name. (Note: To place multiple elements of the same type in a single operation, see "Adding multiple elements" on page 113.)

Tip Alternatively, right-click the diagram background and choose **New** from the right-click menu. The submenu displays all basic elements that can be added to the diagram, and the Shortcuts command.

Configuring element symbol size

When an element is created, its dimensions are set automatically. The maximum and minimum allowed sizes are pre-defined in the view configuration file (`$TGH$/config/view.config`).

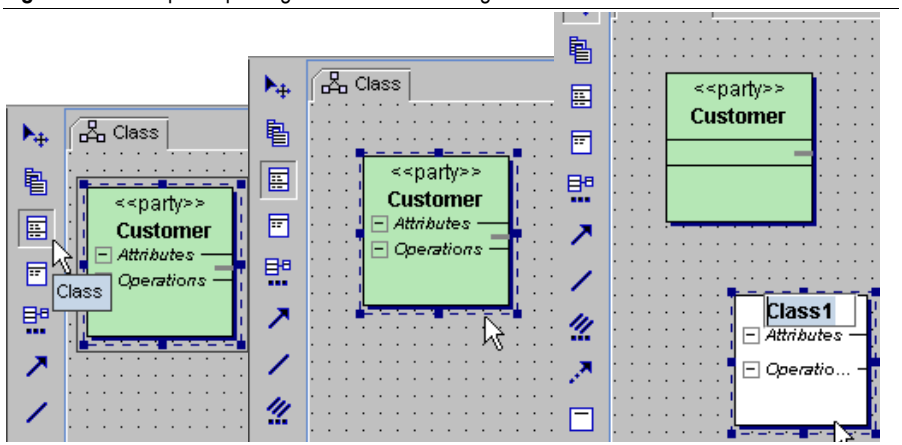
The maximum possible size of an element is configurable in the *Diagram* category of the Options dialog.

To set the maximum size of diagram elements:

1. On the main menu, choose **Tools | Options | <level>- Diagram**.
2. On the Diagram page, choose the field *Maximum width of icons*, and type in the value.
3. Click **OK** to apply the change and close Options dialog.

The size of a diagram element cannot exceed the specified maximum size. If this is the case, the elements are automatically truncated. Thus, the maximum width should not be too small.

Figure 19 Steps for placing an element on a diagram



You can edit the minimum allowed size of a node element in the `view.config` file. If a node is decreased to a smaller size, it will automatically return to the dimensions specified in the view configuration.

If necessary, you can edit the `view.config` file and specify your preferred value of the minimum dimensions.

To set the minimum size of diagram elements:

1. Open the file `$TGH$/config/view.config`.
2. Find the following line:

```
setLayoutConstraints(minWidth(16),minHeight(16),preferredWidth(50),maxWidt
h(maxWidth));
```

3. Type in the new values, and save changes.
4. In the main menu of Together, choose **Tools | Options | Reload changed config files** to apply the changes.

You can resize element symbols by dragging the borders to the desirable size, regardless of the above-mentioned setting. However, you can also return an element to its initial size.

To return an element to its initial size:

1. Select the element on the diagram.
2. On the right-click menu, choose **Layout | Actual Size**. The initial size of an element is restored.

Adding multiple elements

You can place several elements of the same type on a diagram without returning to the toolbar. Each will have a default name that can be edited in place or in the Inspector.

To create multiple elements:

1. Holding down the **CTRL** key, click the toolbar button for the node you want to create (the button stays down). Release the **CTRL** key.
2. Click the desired location on the diagram background. The new element is placed on the diagram at the point you click (the button stays down).
3. Click the next location on the diagram background. The next new element symbol is placed on the diagram.
4. Repeat the previous step until you have one less than the desired number of elements of that type.
5. To place the last element in the series, click once more on the background and close the in-place editor for the element name.
6. Click the **Select** toolbar button or press **ESC.**

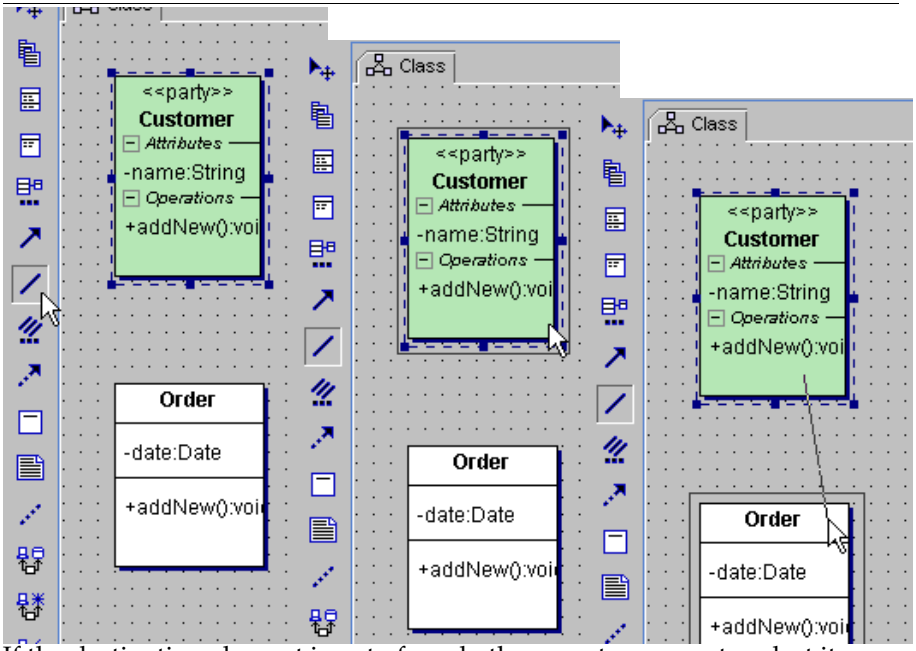
Tip After making a selection on the toolbar or doing the first of a multi-draw or multi-placement operation, cancel the operation by clicking the Select arrow on the toolbar.

Drawing relationship links

To draw a relationship link between two elements:

1. On the diagram toolbar, click the button for the type of link you want to draw in the diagram. The button stays down.
2. Click on the source element.
3. Drag to the destination element and drop when the second element is highlighted, as shown in Figure 20.

Figure 20 Steps for linking elements



Tip If the destination element is out of reach, there are two ways to select it:

- Drag the pointer to the edge of the diagram window so that it scrolls to the desired location.
- Drop the link on the background to open the Choose Destination dialog, and choose the destination element from the list. (Note: To place multiple links of the same type in a single operation, see "Adding multiple elements" on page 113.)

Drawing links with bending points

If your diagram is densely populated, you can draw bent links between the source and target elements to avoid other elements that are in the way.

To create a link with bending points:

1. Click the link button on the toolbar.
2. Click on the source element.
3. Drag the link line, clicking on the diagram background each time you create a section of the link.
4. Click on the destination element to terminate the link.

Tip To configure the links to bend at right angles, choose *Rectilinear* in the Diagram options (**Tools | Options | <level>** - *Diagram: Links*).

Drawing a link to self

On the diagram toolbar, choose the Association link button and double-click the element. This draws a link from the element to itself.

Drawing dependency links

Together automatically shows links between classes and interfaces within the same diagram (for example, if a class extends another class or implements an interface). However, if the diagram contains packages, the dependencies between those packages are not drawn automatically.

The Update Package Dependencies command on the diagram right-click menu builds links between the packages. A link between two packages means that there is a dependency such as implementation, inheritance, or usage. Only actual dependencies are taken into account. For example, if a certain class is imported but never actually used, this is not considered a dependency.

This applies only to source-code based elements. Inter-package dependencies between elements such as objects, notes, search-path imported classes, use cases, and so on are not recognized.

To draw dependency links between packages:

1. Select one or more packages in the diagram.
2. Choose **Update Package Dependencies** on the diagram right-click menu.

Important Together builds all the links between the selected packages and the rest of the diagram. If no particular package is selected, all packages are considered selected. In this case, the command provides two options: *Current Diagram* (self-explanatory) and *All* (be prepared to wait - this builds links for the entire project).

Once built, the dependency links on a diagram are not updated. Obsolete or even non-existent dependencies are displayed until they are deleted or created anew. To update a link individually, select it and choose **Rescan** on the right-click menu for the link.

The process of building dependency links can be time-consuming. However, its speed is configurable. The flag `option.dependencyLinks.fastSearch` in `$TGH$\config\diagram.config` controls recursion into subpackages. Setting this flag to `true` skips subpackages and provides faster operation, while `false` generates the links between subpackages.

Filtering out autodependency links

If a diagram contains autodependency links between the classes, in addition to other links, it can become too complicated. Together provides a special filter for hiding the autodependency links:
`filter.extradependency` in the `%TOGETHER_HOME%/config/filter.config` file.

By default the filter is on and hides all autodependency links if there are any other links between the classes. To change the behavior, edit the `filter.config` file.

Placing shortcuts on diagrams

You can create a shortcut to another diagram or to an element of another diagram. This is useful, for example, when you want to draw an association from a class on one diagram to a class on another. A shortcut to another diagram can reveal a relationship between the two diagrams.

To create shortcuts to other diagrams or diagram elements:

1. Right click the background of the current diagram and choose **New | Shortcuts**.
2. In the resulting dialog, locate what you want to create a shortcut to.
3. Click **Add** to add it to the list of existing or ready to add elements. Repeat this for all the shortcuts you want to make.
4. Click **Ok** to create the shortcut and close the dialog.

Tip You can also create a shortcut to an element by right clicking the element in the Model tab of the Explorer and choosing Add as Shortcut.

A shortcut to a diagram displays as a package that shows the icon of its diagram type and its diagram name. The package lists the elements of the diagram.

A shortcut to a diagram element displays with a shortcut symbol in the lower left corner. (This includes shortcuts to other packages.)

If a shortcut is to a diagram or to a package, you can open the diagram or package represented by the shortcut by double clicking. Alternatively, you can right click and choose Open or Open in New Tab.

Working with elements and links

Most manipulations with diagram elements and link involve dragging the mouse or executing right-click menu commands on the selected elements.

Selecting elements

Click on any element in the diagram to select it.

- To select multiple elements, hold down the **CTRL** key and click on them individually. Alternatively, click on the background and drag a lasso around an area to select all the elements it contains.
- For elements containing members, click on a member to select it.
- Select a node for a diagram element in the Model tab of the Explorer and double-click to select the element in the diagram. Scroll the view to make it visible.

Right-click menus

Many operations on elements or the diagram as a whole can be executed from the various right-click menus. Some of the operations include:

- Add or delete members.
- Set association cardinality. Using the right-click menu of an association link, you can specify the following:
  - Type (association, aggregation, or composition)
  - Cardinality of the client by right-clicking near the client
  - Cardinality of the supplier by right-clicking near the supplier
- Cut, copy, and paste attributes, operations, or text.
- Hide individual elements or show hidden elements.

Right-click on diagram elements, including class members, for access to element-specific operations on the respective right-click menu. Right-click on the background to open the right-click menu for the diagram.

Note The right-click menu of a selected element is duplicated in the main Selection menu.

Tip When right-clicking on an element in a diagram, you may accidentally lose the focus and open the right-click menu of some internal element instead. To avoid this situation, use **Shift+Right-click**. This opens the right-click menu of the desired element and preserves the current selection.

Drag-drop moving and copying elements

You can visually move elements into different packages, or members into different classes or interfaces, by dragging them on top of the destination element and dropping them. For example, you can drag classes into a package (on a class diagram), or components into a node (on a deployment diagram).

Use the same technique to copy members to different classes and interfaces by pressing CTRL before initiating the drag. When you move an element, it is *deleted* from the source package or element and moved into the destination package or element. Copied elements are *not* deleted from their source.

When you drag elements or members around on a class diagram, the appearance of other elements changes when the dragged element crosses their boundaries. This indicates that the symbol being "crossed" has received focus and is a potential drop target. For class, interface, and package elements, the drop focus is represented by a rectangle around the element border. Members are highlighted when focused.

If you drag an element outside the borders of the diagram, the diagram automatically scrolls to follow the dragging.

Full drag-and-drop support

Drag-and-drop support can be extended beyond the borders of a single diagram. This feature is set in the Options dialog - *General: Enable Drag-and-Drop*. When this option is checked, you can drag classes and members between the diagram and the Model tab of the Explorer, or between two diagram tabs. It is even possible to move elements within the Model tab from one node to another. So doing, the elements being moved are deleted from the source component and pasted to the destination component.

Tip This feature changes the behavior of automatic scrolling. The diagram does not scroll when an element is dragged outside the diagram workspace, because the element can be moved to a different component. If you want to scroll the diagram following the dragged element, you have to hold the element for a while near the internal border of the diagram. After a small delay, the diagram resumes scrolling. Note that the link dragging behavior is not affected.

Copying and cloning elements

You can copy and paste elements within the same diagram or between different diagrams. It is also possible to copy and paste members within the same class or between different classes, and to paste a class into another to create an inner class.

- To copy an element, select it and choose **Copy** on the right-click menu or **Edit | Copy** on the main menu.

- To paste an element, select its destination (element or diagram background) and choose **Paste** on the destination's right-click menu or **Edit | Paste** on the main menu.

Note The Cut command works the same way as Copy except the source item is deleted once the Paste operation is complete.

An element that can be cut, copied, and pasted can also be *cloned* using the Clone command on the right-click menu. Cloning is a one-step copy-and-paste.

You can perform cut, copy, paste, and clone operations from the Explorer using the appropriate right-click menu command for an element selected there.

Resizing node elements

To manually resize a node element:

1. Click on the element.
2. Drag the corner of the selected diagram element to the desired size. The element grows or shrinks in the direction of the cursor.

Tip If you hold down the Shift key while dragging the corner of a selected element, the element grows or shrinks uniformly in all directions.

Manually resized nodes shift to an automatically optimized size when their contents change, such as when members are added or deleted.

To restore the default size:

1. Choose **Layout** on the diagram right-click menu.
2. Choose **All** on the submenu.

Converting bidirectional links

An association link between two nodes can be converted to a bidirectional link by using patterns.

To make a link bidirectional:

1. Select the link to be converted.
2. Choose the **Choose Pattern** command on the right-click menu for the link.
3. In the Choose Pattern dialog box, select the **Convert to Bidirectional** pattern, adjust members if necessary, and click **Finish** to complete the operation. This adds the *Split up* checkbox to the list of pattern parameters.

If there are two nodes on a diagram that are associated with each other, both associations can be coupled into a single bidirectional link.

To combine two associations in a single bidirectional link:

1. Hold down the **CTRL** key and click on each link.

2. Choose the **Choose Pattern** command on the right-click menu of the selection.
3. In the Choose Pattern dialog box, select the **Convert to Bidirectional** pattern, adjust members if necessary, and click **Finish** to complete the operation. This adds the *Split up* checkbox to the list of pattern parameters.

All common operations are applicable to bidirectional links as well: you can move, reroute, or delete them as required. If a bidirectional link is not needed, revert to two single associations.

To split one bidirectional link into two separate associations:

1. Select the bidirectional link.
2. Choose the **Choose Pattern** command on the right-click menu of the selection.
3. Check the *Split up* box and click **Finish**. The link breaks apart into two associations.

Note If one (or both) of the associated classes is read-only, the Bidirectional Link pattern cannot be applied.

Labeling links

You can add textual labels to links to add information to the model. Three labels can be associated with a single link: the link label itself, a client label, and a supplier label.

To add a link label:

1. Choose **Rename** on the right-click menu of the link.
2. Choose the **Label** command from the submenu to open the in-place editor for the link label.
3. Enter the link description and press **Enter**.

To add a client/supplier role label:

1. Choose **Rename | Client Role (Supplier Role)** on the right-click menu of the link. This opens the in-place editor for the role.
2. Enter the role description and press **Enter**.

The maximum length of the description is specified in the Options dialog (*Diagram: Maximum link label length without wrapping*). If the description is too long, the text is displayed in multiple lines, but only if this feature is enabled in the Options dialog.

To enable label text wrapping:

1. Choose **Diagram Options** on the right-click menu of the diagram.
2. Select the *Diagram* node.
3. Check the *Wrap link label text* box and close the dialog.

Labels can be oriented along the links. This behavior is controlled by the option *Show labels oriented along links* on the Diagram node of the Options dialog. However, oriented labels and multi-line labels are mutually exclusive. If oriented labels are selected, word wrapping is disabled.

Changing links

Once the links are created, you can change the client or supplier nodes of a link, add or remove bending points, and reverse link direction.

To change the link routing:

1. Select the link.
2. Move the mouse cursor to the source or destination point of the link. As the mouse cursor hovers over the target point, its shape changes to .

Tip If the required end of the link is out of reach, choose **Scroll to Source** or **Scroll to Destination** on the right-click menu of the link.

3. Drag either the source or destination end point of the link to a new element.

Tip If the drop destination is out of range, drop the link end point on the diagram background and select the target in the dialog that opens.

To add bending points to a link:

1. Select a point on the link. As the mouse cursor hovers over the target point, its shape changes to cross-hairs.
2. Drag the selected point and drop in the required place on the diagram background.

To remove bending points:

1. Select the client or supplier node element.
2. Choose **Layout | Route Links** on the right-click menu.

To reverse a link:

1. Select the link.
2. Choose **Reverse Link Direction** on the right-click menu.

Viewing diagrams and managing diagram layout

Together makes it easy to manage simple or complex diagrams with automated layout features that optimize the diagram layout for viewing or printing. You can also create your own diagram layouts.

Viewing

You can scroll the Designer pane horizontally and vertically using its scrollbars or the Overview button in the lower right corner of the Designer pane.

- Resize the pane vertically by dragging its lower edge up or down.
- Resize the pane horizontally by dragging the separator located between the toolbar and the Explorer pane or by resizing the window.
- Enlarge your work area in the Designer pane by hiding any or all of the other panes using the View menu or main toolbar buttons.
- Click the Overview button (or choose **View | Diagram | Overview**) to display the overview of the current diagram. Resize and move the shadow area to obtain the desired view.

Zooming

There are several alternative ways to obtain the required magnification on the Designer pane.

- Use the Zoom lens button on the diagram toolbar:
  - Click the Zoom lens button and click on the Designer pane to zoom in.
  - Hold down the **Alt** key and click on the Designer pane to zoom out.
- Choose **View | Zoom** on the main menu and choose a command from the submenu.
- Use keyboard shortcuts. Default settings use numeric keypad keys:
  - NumPad+ for Zoom In
  - NumPad- for Zoom Out
  - NumPad* for Fit in Window
  - NumPad/ for Fit 1:1

Using the automated layout features

The diagram right-click menu provides access to the automated layout optimization features with the following commands.

If you click on the background:

- **Layout | All** positions all diagram elements automatically according to the Layout options settings.
- **Layout | All for Printing** positions elements within page borders. You can set Print options to display the print grid on your output.

If you select one or more diagram elements:

- **Layout | Selected** and **Layout | Selected for Printing** reposition only selected elements.
- **Layout | Route Links** streamlines the links.
- **Layout | Actual Size** optimizes the element size.

Important The contents of the Layout submenu depend on the current selection.

Creating your own manual layout

You can create your own layout by selecting and moving single or multiple diagram elements:

- Select a single element and drag it to a new position.
- Select multiple elements and drag them as a group to a new position.
- Select multiple elements, cut them as a group, and paste them to a new position.
- Manually reroute links.

Tips and tricks

- Manual layouts are saved when you close a diagram or project and restored when you next open it.
- Manual layouts are *not* preserved when you run one of the auto-layout commands.
- You can revert to your manual layout after an auto-layout operation using Undo. For example, you might invoke Layout All for Printing, print the diagram, then call Undo to restore your manual layout.

Editing diagrams

After opening a project that has diagrams, you can open one or more diagrams for editing. Each diagram opens in its own tabbed page in the Designer pane. Tabs display the diagram name and diagram type icon. (The icons correspond to those shown in the Model tab of the Explorer.) Switch between open diagrams by clicking the desired tab.

Opening and closing diagrams

Diagrams are opened from the Model tab of the Explorer using the right-click menu commands.

To open a diagram:

1. In the Model tab of the Explorer, navigate to the diagram you want to open.

2. On the right-click menu of the diagram, choose **Open** to replace the contents of the current diagram, or **Open in New Tab** to open the diagram without closing the current one.

Tip Following are alternative ways to open diagrams:

- Double-click on the diagram node. This opens the diagram in the Designer pane, replacing the diagram currently in focus.
- On the right-click menu of a package, choose **Open Diagram** or **Open in New Tab**.

Diagrams are closed using the commands on the main menu or diagram right-click menu.

To close the current diagram:

1. Right-click on the diagram tab.
2. Choose **Close**.

Tip Following are alternative ways to close current diagrams:

- Use one of the keyboard shortcuts, Ctrl+F4 or Ctrl+W.
- Choose **File | Close** on the main menu.

To close all opened diagrams:

1. Right-click on the diagram tab.
2. Choose **Close All**.

Editing properties

When you select an element in a diagram or the diagram background, the properties of the element (or diagram) are available in the Inspector. The contents of Inspectors vary according to the selection. If nothing is selected, the diagram background is selected by default and diagram properties are displayed.

Inspectors have several tabbed pages representing different categories of properties. For example, the Inspector for a class has the following tabs:

Properties. General properties of the class (name, stereotype, abstract, and so on). This tab displays two columns:

- **Name** shows the name of each property.
- **Value** displays the value or setting of the property named in the same row of the *Name* column.

Hyperlinks. Intra-project and URL hyperlinks to related information (for information on hyperlinking, see "Hyperlinking diagrams" on page 127).

View. Visual properties of the diagram symbol, such as color.

Description. Source code comments.

Javadoc. Properties used for Javadoc generation, such as author and version.

HTMLdoc. Source code comments in HTML format.

Requirements. Information tracking requirements, such as number, priority, and difficulty.

Beans. Optional JavaBean properties, if the class is a JavaBean.

Stereotype. The *stereotype* property typically has a pick-list but these are not always populated by default. You can modify an Inspector to add or modify stereotypes with some Java programming. For information on adding or modifying stereotypes, see "Customizing Property Inspectors" on page 453.

In general, you can enter a value for string type properties by typing a value in the edit field of the *Value* column. Some string type properties display a browse button next to the field to indicate that an editor dialog is available.

- Multi-line string properties display a multi-line text editor dialog.
- String properties whose value must be the name of some object or element in the project display the file chooser button.

Tip Use the keyboard shortcut Alt+Insert to edit the fields with file chooser buttons.

- Some properties can have multiple values, such as *parameter* of an operation. Here, the editor accepts a comma-delimited string.
- Some properties have a pick-list of available values. You can select from the list or enter your own value in such fields.

Opening the Inspector

There are several ways to open the Inspector:

- Choose **Properties** on the right-click menu.
- Choose **Selection | Properties** on the main menu.
- Use the keyboard shortcut Alt+Enter (or Alt + double-click).

Modifying properties

To modify a property value, click it in the Inspector. Some properties are not modifiable with the Inspector and are therefore disabled for editing. Note also that some diagram elements, such as compiled classes, are read-only and therefore all their properties are disabled for editing.

Applying changes

Changes to property fields are applied when you:

- Press **Enter** (Inspector dialog remains open).
- Close the Inspector (Ctrl+Enter, Alt+F4, Esc, or click the Close button).
- Exit the edited field.

- Lose the focus on the open Inspector dialog.

Important If you click the Close button in the dialog while changes are pending to a field, *the changes are not saved.*

In-place editing

In addition to editing properties of elements in the Inspector, you can do in-place editing on the diagram itself. Every element on the diagram has an identifying string with a certain set of properties. Some of these properties are not displayed and become available only during the course of in-place editing. You can modify all properties or just some of them, leaving the underlying engine to complete the changes.

There are several ways to enable in-place editing:

1. Double-click the selected element.
2. Choose **Rename** on the right-click menu for the element.
   - or -
   Press **F2** for the selected element.

   This opens a highlighted text string with a cursor for modification, unless the diagram is read-only.

To apply changes and thus generate the relevant code:

1. Press **Enter** (except for Note elements)
2. Click on another element

Note When editing Note elements, keep in mind that Enter creates a new line and Ctrl+Enter applies editing and closes the in-place editor.

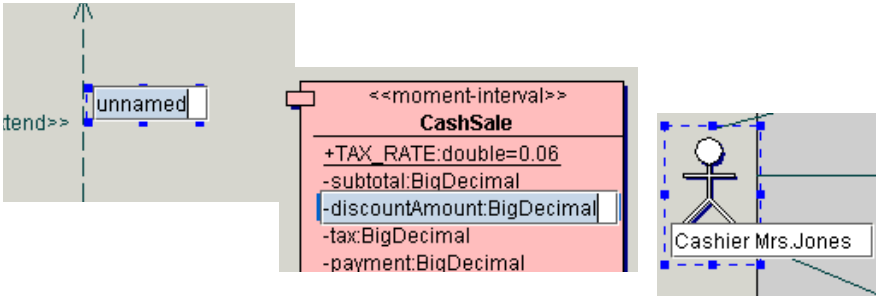
On code-generating diagrams, you must adhere to correct syntax when editing properties manually. Together responds to the wrong syntax of modifiers with error messages. But beware - a misspelled attribute type on a diagram is reproduced exactly in forward engineering.

In case of an incomplete entry, the omitted visibility modifiers, attribute types, or return types of the operations are replaced with their existing values. For example, if an attribute was private and the visibility is not entered, then it remains private. If no value was specified before, then default values are substituted as they are specified in the appropriate template properties. For example, an attribute with undefined visibility modifier and type will be a `private int` and if the return type of an operation is not specified, it defaults to `void`.

Most strings can be modified at any time.

Note Shapetypes and link types are not editable.

Figure 21 In-place editing of a link comment, member, and actor



Hyperlinking diagrams

This section explains the types of hyperlinks that can be created with Together and their benefits. There are also step-by-step instructions for creating, viewing, removing, and browsing hyperlinks.

The hyperlinking feature enables you to:

- Create hyperlinks from your diagrams to other system artifacts and browse directly to them. Create hyperlinks from the current diagram as a whole or from a selected diagram element (or group of elements) to:
  - A new diagram.
  - An existing diagram or diagram element anywhere in the project.
  - A document file on a local or remote storage device.
  - A URL on your company intranet or on the Internet.
- Create, view, remove, and browse hyperlinks using the right-click menu on the Hyperlink tab in the Inspector.
- Add or remove hyperlinks using the right-click menus of the link-type nodes of the Hyperlinks tab.

Note Hyperlinking is not available for some types of links between diagram elements. For example, generalization/implementation links do not allow hyperlinking. The Hyperlink to command is not provided on the right-click menu of such links. For association links, hyperlinking is allowed to files and URLs only.

Why use hyperlinking?

- Link diagrams that are generalities or overviews to specifics and details.
- Create browse sequences leading through different but related views in a specific order; create hierarchical browse sequences.
- Link descendant classes to ancestors; browse hierarchies.

- Link diagrams or elements to standards or reference documents or generated documentation.
- Facilitate collaboration among team members.

How to create hyperlinks

This section explains how to create various kinds of hyperlinks. Hyperlinks exist within the context of projects, so open a project to create or browse them. Hyperlinks are *properties,* which is why they appear in the Inspector. You can create hyperlinked diagrams using one of the following possible tools:

- Hyperlinks tab in the Inspector.
- Hyperlink To submenu on the Selection menu.
- Right-click menu of the selected element or current diagram.

Hyperlinking to a new diagram

You can create a hyperlink from an existing diagram or one of its elements to a new diagram that you create as part of the hyperlinking task. The new diagram is hyperlinked to your originating element by default.

For instructions on hyperlinking to a new diagram, see "Using the hyperlink feature to create diagrams" on page 108. The new diagram opens in the Designer pane and the Hyperlinks tab in the Inspector displays the link to the originating diagram or element.

Hyperlinking to an existing diagram or diagram element

Create a hyperlink from an existing diagram or one of its elements to any other diagram or diagram element in the project.

To create a hyperlink to an existing diagram or element:

1. Open an existing diagram from which to create the hyperlink (or create a new diagram).
2. Select the element or group of elements that you want to link to another diagram or element. To link to the diagram as a whole, click on the diagram background.
3. On the Selection menu, choose **Hyperlink To** | **Existing Element**. The Select Element dialog opens.
4. Choose the desired diagram or element in the dialog and click **Add**. To find an element, expand diagram nodes in the tree view.
5. Click **OK** to close the dialog and create the link.

Hyperlinking to a URL or file

You can create hyperlinks from your diagrams to any online resource. For most users, such hyperlinking will probably take the form of documents on a LAN or document server, or URLs on the company intranet. But you can just as easily link to online information from the OMG, newsgroups, or discussion forums. If it is available online, you can link to it.

Note that file paths or URLs can be absolute or relative to the project.

To create a hyperlink to a file or URL:

1. Open an existing diagram or create a new diagram.
2. Select the element or group of elements to link to another diagram or element. To link to the diagram as a whole, click on the diagram background.
3. On the main menu, choose **Selection | Hyperlink To | File/URL**. The Choose URL dialog opens.
4. Choose the type of URL: **Project Relative** or **Absolute**.
5. Type in the complete path or browse for it. Note that if you change between relative and absolute path, then the specified path changes accordingly.
6. Click **Ok** to create the link.

Viewing hyperlinks

All the hyperlinks defined for any diagram or element are displayed on the Hyperlinks tab of the Inspector. Select the Hyperlinks tab before checking diagrams for defined hyperlinks.

To view hyperlinks to a diagram, click on the diagram background and then view the Hyperlinks tab for defined links.

On a diagram, all names of diagram elements that are hyperlinked are displayed in blue font. To view what is hyperlinked to the element, click on the element (node or relationship link) and then view the Hyperlinks tab.

Browsing hyperlinked resources

Once you find the defined hyperlinks for a selected diagram or element, use the right-click menu on the Hyperlink tab to browse to the linked resources.

- Browsing to a linked **diagram** opens it in the Designer pane or makes it the current diagram, if already open.
- Browsing to a linked **element** opens its parent diagram, makes it current, and scrolls the diagram to the linked element and selects it.
- Browsing to a linked **file** launches the application registered in the system for the file type and loads the file.

- Browsing to a linked **URL** launches the default system Web browser and loads the URL.

To browse a hyperlink:

1. Right-click the desired hyperlink on the Hyperlink tab.
2. Choose **Open** on the right-click menu.

Removing hyperlinks

Hyperlinks can be removed from either "end" of a diagram or element hyperlink. That is, if you created a hyperlink from the diagram *Foo* to the diagram *Bar,* you can remove the hyperlink from either *Foo* or *Bar*.

To remove a hyperlink:

1. Open the diagram that has the link you want to remove.
2. Choose **Properties** on the diagram right-click menu.
3. Right-click on the link in the Hyperlinks tab.
4. Choose **Remove** from the right-click menu.

Annotating diagrams

The Diagram Elements toolbar for each diagram type has Note and Note Link buttons for placing notation on a diagram. Notes can be free floating, or you can draw a note link to some other element to show that a note pertains specifically to it.

Using Notes

- Paste text from the clipboard into a note when its in-place editor is active.
- Note text wraps to the next line when you resize the note.
- Edit the properties of a note using its Inspector (Alt+Enter).

In the Note Inspector you can:

- Edit the text as text, inserting HTML tags on the Text tab if desired.
- Change the text-only property on the Properties tab.
- View the note text as HTML on the HTMLdoc tab.
- Change the foreground and background colors, and control 3D appearance on the View tab.
- Track various requirements properties including stereotype, priority, and difficulty on the Requirements tab.

- Add custom properties on the Custom Properties tab.

Note Links

Notes are automatically included when you generate HTML documentation. The text of notes linked to class diagram elements does not appear in the source code.

To open the Inspector for a note link, select Properties from the note link right-click menu. The Inspector for a note link is similar to the note Inspector, but the Text tab is replaced by the Link tab. On the Link tab you can view both ends of the link - client and supplier.

Inspector documentation tabs

The Description tab in element Inspectors displays the description of the selected element. If the element is in a source generating diagram, the text in this tab becomes a Javadoc comment in the source code corresponding to the element. When you edit the text in this tab, the source code is updated when you press Alt+Enter or return to the diagram.

You can use the Description tab to create and edit comments for notes or elements on diagrams that do not generate code. These comments are stored with other diagram-specific information.

Use the Javadoc tab to enter the values of Javadoc tags for source code comments.

In addition to the tabs for viewing and editing comments as plain text, the HTMLdoc tab displays comments as formatted in a browser.

Standalone design elements

By default, design elements are stored inside the diagram packages. However, this simple organization makes it difficult to share elements among team members or store the elements in a source code control system.

To handle these problems, Together enables storing design elements in separate files. Such design elements are called Standalone Design Elements (SDEs). Physically, they are stored in files with the `.ef<shapetype>` extension. For example, a standalone actor can be stored in a file named `Actor1.efActor`.

Important A design element is any element that has no underlying source code such as an actor, an object, a swimLane, or a note, but not a class or a package.

Creating standalone design elements

To create SDEs:

1. On the main menu, choose **Tools | Options | <level>**.
2. In the *General* options group, check *Create design elements as standalone.*

When this option is checked, any new design element created on a diagram is added as a shortcut, and the appropriate `.ef<shapetype>` file is written to the physical package of this diagram.

Note On the Directory tab of the Explorer, the icons for the standalone design elements are different than the icons for ordinary design elements.

There is an alternative way to create SDEs when this option is unchecked: create a design element on a diagram, copy it, and add it to the model as an SDE.

1. In the Designer pane, copy an element.
2. On the Model tab of the Explorer, select the destination package and right-click.
3. Choose **Paste as Standalone** on the package right-click menu.

Tip If a diagram contains existing non-standalone design elements of the same shapetype, their names are not taken into account when new names for the new SDEs are assigned. However, all the standalone elements have different names assigned following the same rules as the names of the non-standalone design elements.

Using standalone design elements

As mentioned above, with the *Create new design element as standalone* option selected, each subsequent design element added to a diagram is created as a separate file in the diagram package. You can observe all different files on the Directory tab of the Explorer, such as `Actor1.efActor, Actor2.efActor, Actor3.efActor`.

If this option is unchecked and you still want to make use of an existing or imported standalone design element, perform the following steps:

1. Select the standalone element in the Model tree-view.
2. Choose **Copy** from the node right-click menu.
3. Right-click on the target diagram and choose **Paste** or **Paste Shortcut**.

Saving and copying diagram images

You can save or copy entire diagrams or their selected parts for further reuse. There are three possible behaviors: copying and pasting within Together, copying the image, and saving the image.

Copy - paste within Together

1. Select the required part of a diagram.
2. Choose **Edit | Copy** on the main menu or the right-click menu.

3. **Paste** the selection to the target location.

Note If link labels are not displayed after pasting, press F5 with the focus on the Designer pane to refresh the diagram.

Copy image

In the Windows environment, you can copy a diagram to the clipboard and then paste the clipboard content to an external application.

Choose the format for copying images: bitmap (BMP) or Windows Metafile (WMF) format. Bitmap format solves several problems with WMF format such as distorted fonts, incorrect conversion of localized fonts, and lack of Java 2D functions support. In addition, some rasterized graphics applications do not handle clipboard metafiles, but they do recognize bitmaps.

To set the format for copying diagrams, open the Options dialog and check or uncheck the option *General: Copy diagram to clipboard as bitmap*.

Note When a diagram is copied in WMF format, labels of non-horizontal links are not reproduced in the target application.

Tip Pasting bitmap images to MS PowerPoint produces "invisible slides." To avoid this, use the Paste Special command, rather than Paste. However, if you copy a diagram in WMF format, it can be reproduced in PowerPoint using the Paste command.

Save image

A diagram can be saved on the hard disk for further use. Together offers a choice between WMF, GIF, and SVG formats. To save the current diagram, choose the **Save Image** command on the File menu, and select the required target format on the submenu. This opens the Save Diagram dialog for the selected format.

The diagram image is stored on the disk and can be imported to applications that recognize the selected format.

Printing diagrams

You can set printing options at different levels in the Options dialog.

To check Together Print options for the specified level:

1. Choose **Tools | Options | <level>** on the main menu to open the Options dialog for the default, project, or diagram level.
2. Select the *Print* node and check the settings.

For further information, see Chapter 2, "Setting Your Personal Preferences" on page 49.

How to print diagrams

You can print diagrams separately or as a group, or print all diagrams in the project.

To print a single diagram:

1. Open the diagram in the Designer pane (the project must be open). If there are several open diagrams, click the tab of the one you want to print.
2. Choose **File | Print Diagram** on the main menu to open the Print Diagram dialog.
3. By default, the scope is set to *Current*. Click **Print**.

To print several diagrams at once:

1. Open the diagrams in the Designer pane (the project must be open).
2. Click on the workspace of any open diagram.
3. Choose **File | Print Diagram** on the main menu to open the Print Diagram dialog.
4. Choose *All opened* and click **Print.**

Using auto-layout for printing

Together has automated layout optimization for printing diagrams. Auto-layout for printing ensures that all diagram elements fall within page borders defined by page size in Print Options.

Invoke print auto-layout immediately before printing a diagram. Right-click on the diagram background, then choose **Layout | All for Printing**.

Tip To preserve a manual diagram layout, invoke Undo after running auto-layout and printing the diagram.

Printing generated documentation

- After generating HTML documentation, open it in your Web browser and print from there.
- After generating documentation in another format, open it in an application that reads the file format and print from there.

Troubleshooting

Printing problems (such as distorted diagram images) can stem from a number of things unrelated to a particular application program such as Together.

If you are working on a UNIX system, check the Known Problems section in the `readme.html` file in the root of your Together installation. There are specific printing issues for Together on some UNIX systems.

Under Windows, Together provides a workaround for printing via the property: *print.dpi=72* in the `$TOGETHER_HOME$/config/misc.config` file. This property prevents problems in most cases.

If you experience printing problems, try one of the following:

- Adjust the value of the `print.dpi` property.
- Comment out the `print.dpi` property.

Then try printing again.

For each test:

1. Modify and save the `misc.config` file.
2. Assuming Together is running, choose **Tools | Options | Reload Changed Config Files**.
3. Print the diagram (**File | Print Diagram**).

Tips and tricks

- If you are running Together under Windows, set the True Type Fonts property for your printer to *Print As Graphics* in the Font tab of the Windows Printers dialog before printing diagrams.
- To make sure that all diagram elements fall within page boundaries, run the auto-layout command Layout All for Printing from the diagram right-click menu before printing a diagram. If you have created a manual layout, restore it using the Undo command after printing. Be sure to run Undo *before closing the diagram* or your manual layout will be lost.

Chapter

9

UML Modeling

Together provides support for the most frequently used diagrams and notations defined by the Unified Modeling Language (UML). For thorough discussions of the UML, refer to the Object Management Group website at *http://www.omg.org*.

This chapter includes the following topics:

- "Notation and stereotypes of UML diagrams" on page 137.
- "Use case diagrams" on page 138.
- "Class diagrams" on page 140.
- "Sequence and collaboration diagrams" on page 147.
- "Statechart diagrams" on page 156.
- "Activity diagrams" on page 159.
- "Component diagrams" on page 162.
- "Deployment diagrams" on page 164.

Notation and stereotypes of UML diagrams

Together supports both notation and stereotypes of UML diagrams:

- **Notation**

  Together enables you to render UML diagrams using UML-compliant notation. In class diagrams, you control the code generation of various notational elements through either global or diagram level configuration options.

- **Stereotypes**

Together supports the use of stereotypes. You can adhere to UML-defined stereotypes, or customize stereotypes based on your requirements. You can also add color to stereotypes, although this practice is not currently specified in the UML.

Use case diagrams

Use case diagrams describe the external view of a system and its interactions with the outside world. Actors are entities outside the system that interact with it. A use case is an action or sequence of actions required by the system. A use case diagram depicts the relationship between actors and use cases within a system.

Figure 22 is an example of a use case diagram.

Figure 22 Use case diagram

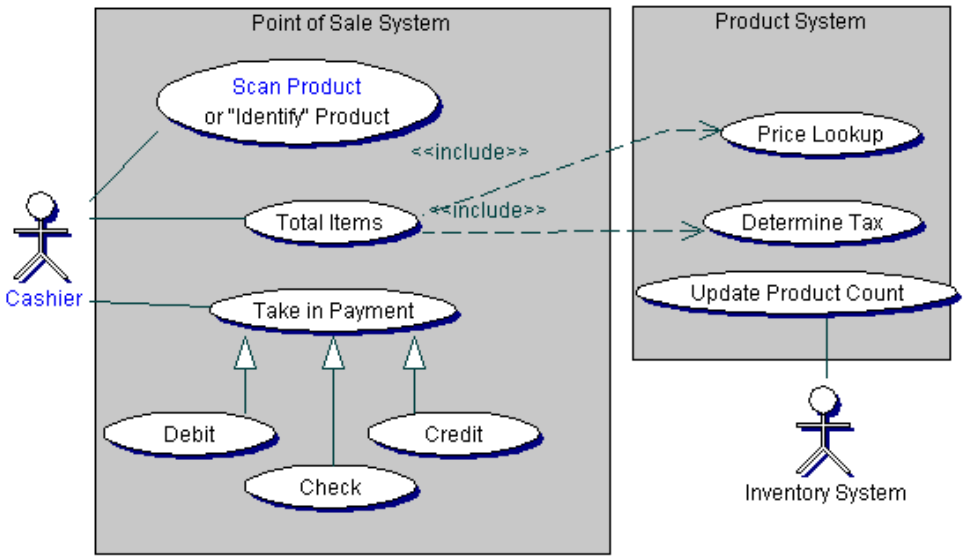


Diagram elements

Table 12 lists the elements of use case diagrams that are on the diagram toolbar. Relationship links are in *italicized* text.

Table 12 Elements of use case diagrams

Icon	Element name
	Actor
	Use Case

Table 12 Elements of use case diagrams (continued)

Icon	Element name
	Communicates
	Extends
	Includes
	Generalization
	System boundary
	Note
	Note link

To learn how to create new diagrams in a project or to understand the techniques for placing elements and drawing links, see "Working with Diagrams" on page 107.

Creating browse-through sequences of use case diagrams

When you develop use cases, you typically begin at a high level and specify the main use cases of the system. As you proceed, you break these into finer detail. For example, a "Conduct Business" use case may have another level of detail that includes use cases such as "Enter Customers" and "Enter Sales." It is useful to have a method of expanding or contracting the use cases to view the system at different levels of detail.

Together's hyperlinking feature enables you to create browse-through sequences comprised of several use case (or other type) diagrams. You can link diagrams at one level of detail to the next diagram up or down in a sequence of increasing granularity, or you can link from key use cases or actors to the next diagram. By browsing the hyperlink sequence, you can follow the relationships between the use case diagrams.

You can use hyperlinking to link diagrams and elements based on your requirements. For example, you can develop a hierarchical browse-through sequence of use case diagrams, creating hyperlinks within the diagrams that follow a specific actor through all use cases that reference the actor.

To learn how to create and browse hyperlinks, see "Hyperlinking diagrams" on page 127.

Note You can also show that two use case diagrams are related by placing a shortcut of one on the other. See "Placing shortcuts on diagrams" on page 116.

Adding extension points

An extension point is a location within a use case where you can insert action sequences from other use cases. Each extension point in a use case has a unique name. The list of extension points is ordered according to the behavior of the use case.

In a use case diagram, extension points are listed in the use case with the heading **Extension points** (appears as bold text in the diagram).

To add an extension point to a use case:

1. Right-click the use case and choose **New Extension Point**.
2. Edit the extension point in place.

The new extension point is added at the end of the list of existing extension points. If necessary, drag the extension point to change its position in the list.

Class diagrams

Class diagrams show the static structures of systems. This includes classes and interfaces and their internal structure as well as relationships between classes and inheritance structures, as shown in Figure 23. A class diagram can also show objects and links between objects.

Figure 23 Class diagram

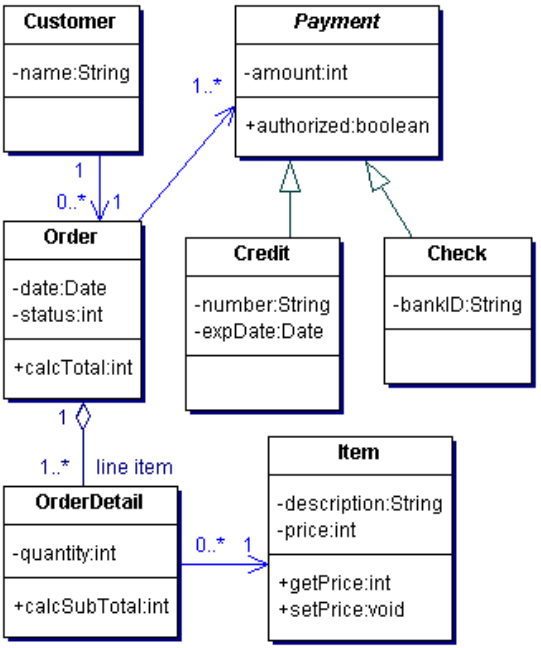


Diagram elements

Table 13 lists the elements of class diagram that are on the diagram toolbar. Relationship links are in *italicized* text.

Table 13 Elements of class diagrams

Icon	Element
	Package
	Class
	Interface
	Class by pattern
	Generalization/Implementation
	Association
	Link by pattern
	Dependency
	Object

Table 13 Elements of class diagrams (continued)

Icon	Element
	Note
	Note link
	Entity EJB
	Session EJB
	Message-driven EJB

In addition to these elements, class diagrams can contain shortcuts to the elements from other packages and diagrams.

Physical (package) and logical class diagrams

The Model tab of the Explorer uses two class diagram icons:

- Physical class diagrams, which are referred to in this chapter as *package diagrams*. Package diagrams are stored as a text files with the extension `dfPackage`.
- Logical class diagrams. These diagrams are stored as text files with the extension `.dfClass`.

Together creates a package diagram for the project (`<default>` diagram) and for each subdirectory under the project directory. The subdirectory paths are defined in the project properties (Project | Project Properties). You must create any logical class diagram separately.

Working with logical class diagrams

You can create a new logical class diagram in the usual manner, as described in "Creating diagrams in projects" on page 107. When you place a class, interface, or package on a class diagram, Together generates the corresponding source code or subpackage in the package that you specified when you created the diagram.

The impact of changing a class, interface, or package on a logical class diagram varies according to the kind of change:

- Changing the name, adding a member, creating a new link, or applying a pattern makes the corresponding change in the actual source code.
- Dragging a class to a package on the logical class diagram either copies the class to the package or creates a shortcut to the class in the package, depending on your choice. It does not change the location of the actual corresponding class.
- Removing a class, interface, or package from a logical class diagram does not remove it from the project or from its corresponding package diagram.

Working with package diagrams

When you open a project subdirectory from the Model tab of the Explorer or from the diagram, Together reverse engineers the contents into a package diagram that shows the packages, classes, and interfaces and their interrelationships.

Opening packages

To open a package, choose one of the Open commands on the right-click menu of the package in the Model tab of the Explorer.

Deleting packages

To delete a package from a package diagram or from the Model tab of the Explorer, choose Delete from its right-click menu. Deleting a package deletes all of its contents.

Renaming packages

To rename a package, including changing the package name in all of its source files, choose Rename from its right-click menu. You can also rename by using the package's Inspector (Right-click menu | Properties).

Viewing packages and their content

By default, a package element on a diagram displays the package contents. You can use the right-click menu of a class or interface in a package to add attributes and operations directly.

To show or hide the contents of package elements:

1. Right click the diagram background and choose **Diagram Options** to open the Diagram Options dialog.
2. Select **View Management | Show subpackage contents**.
3. Check or clear *Show subpackage contents* as desired.

Moving and copying elements from packages

To copy packages, classes, or interfaces from another project package to a class diagram:

1. In the Model tab of the Explorer, navigate to the package containing the element(s) you want to show.
2. Locate the desired node in the Explorer and copy to the clipboard. Then paste from the clipboard to the desired location.
3. After completing the copy operations, choose **Update Package Dependencies** on the diagram right-click menu to update any package dependency links as desired. (For an explanation of updating package dependencies, see "Drawing dependency links" on page 115.)

To move a class, interface, or package into or out of a package on the diagram, use one of the following methods:

- By drag and drop using the left button of your mouse. Use the right button to display the **Move** command.
- By copy and paste. In addition to working in the Designer pane, this method also applies to the Explorer pane.

Showing classes on search paths

Projects can have search paths whose content you may want to display in diagrams. For example, you can show entities that reside on the standard libraries or the full Java classpath. Such resources exist for the project, but Together does not include them in the generated HTML documentation for the project.

To show classes from a search path in a class diagram:

1. Open or create a class diagram.
2. Right-click on the diagram background and choose **New** | **Shortcuts**. The resulting dialog displays the content available for the diagram and all content residing outside the current package.
3. Select the resource you want to add and click the **Add** button. Repeat until you have added all the resources you want.
4. Click **Ok** to close the dialog.

Tip If the resource you are looking for is not shown, it is probably not in the search paths defined in Project Properties. For instructions on adding resources to project search paths, see "Adding Resources" on page 79.

Showing JavaBeans

Together can recognize classes that match the bean pattern as JavaBeans.

To show classes as JavaBeans on a class diagram:

1. From the main menu, choose **Tools** | **Options** | **<level>** to open the Options dialog.
2. From the tree, choose **View Management** | **JavaBeans**
3. Check the *Recognize JavaBeans* option.

Any class that satisfies the bean pattern shows a small rectangle at the top of the left side. All properties and events display in separate compartments below the operations compartment.

Defining inner classes

To create an inner class from a class that already exists, drag the existing class to the outer class and drop it. If the inner class does not exist, you can create and define it within its outer class at the same time.

To create and define a new inner class:

1. Select the outer class.
2. Choose **New** | **Inner Class** from the right-click menu.

Tip You can also define new inner classes from the right-click menus of class nodes in the Explorer.

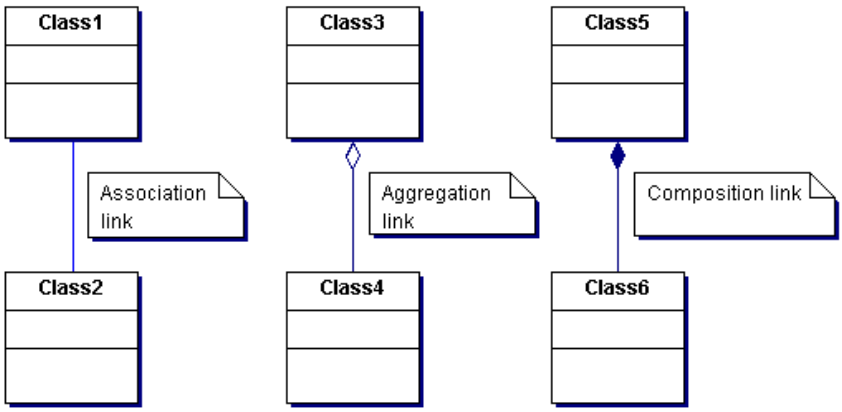
You can also use drag-drop to remove inner classes from classes in the diagram.

Creating associations, aggregations, and compositions

Use the association button on the class diagram toolbar to draw association links between diagram elements. Right-click on an existing link to specify an aggregation or composition link. You can change the code corresponding to the link by applying the appropriate pattern to the link.

Figure 24 illustrates how Together depicts the three types of association links.

Figure 24 Types of association links



When you create an association link, Together defines an attribute in the client class (the start of the link) named `lnkClass`, where `Class` is the name of the target class. By default, such attributes are not displayed in the class elements.

The right-click menu of a link enables you to set the link type (association, aggregation, composition) and the cardinality of the client and supplier. If you right click the link near the supplier, you can change its cardinality. Right click the link near the client to change its cardinality.

Associations can be automatic, directed, or undirected. A directed link points to the supplier class (the target). An automatic link points only if the name of the corresponding attribute in the client class does not begin with `lnk`.

To set the directed property of an association link:

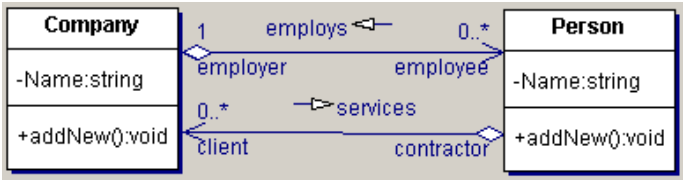
1. Right-click on the link and select **Properties** to open the Inspector.

2. From the Link tab of the Inspector, set the *directed* property to **Automatic**, **Directed**, or **Undirected**.

You can show the semantic direction of a link by specifying a label and label direction in the link Inspector.

Semantic direction may be independent of the link navigation direction, as shown in Figure 25. In this example, possible relationships exist between Company and Person. Note that the label direction and semantics depict that Company employs Person in the first instance, and Person services Company in the second case.

Figure 25 Semantic direction of the link label vs. link direction



Editing members and properties

This section includes instructions for adding and editing members and properties of classes and interfaces.

Adding and editing members

To add an attribute or an operation to a class or interface:

1. Right-click on the class or interface
2. Choose **New | Attribute**, **New | Operation**, or **New | Constructor**.
3. Edit the member with the in-place editor, or use its Inspector.

The new member is inserted at the end of its compartment in the class element on the diagram.

If a class already has attributes or operations, you can right-click on one of them to create an additional member using the right-click menu. The new member is inserted before the selected member.

Tip Class and interface nodes in the Model tab of the Explorer have right-click menus for adding new members.

Adding and editing properties

To add a property to a class, right-click on the class and choose **New | Property**.

The location of the newly created property depends on whether the Recognize JavaBeans option is on. If so, the new property is added to the properties compartment and to the list of properties in the Beans tab of the Inspector. The class displays as a bean symbol. If the option is off, the property is added to the attributes section, and its accessor methods are added to the operations section.

When the Recognize Java Beans option is off, you must take special care when editing or deleting properties. If you edit a new property with the in-place editor, the relevant types for the accessor methods will not be synchronized. When a property is deleted, the accessor methods remain and you must delete them individually. The safest method for editing properties is to use the Choose Pattern dialog.

To edit properties using the Choose Pattern dialog:

1. Right-click on the property to be edited and select **Choose Pattern**.This opens the Choose Pattern dialog.
2. Choose the **Property pattern** and complete all necessary changes.
3. Click **Finish** to apply the changes to all the components of a property.

Rearranging the order of attributes or operations

You can use drag and drop to reorder members within a class. Together simultaneously updates the source code with the new order.

Dropping one member on the name of another member positions the dropped member before the target member. Dropping a member on the class name moves it to the last position in the attribute or operation list respectively.

Setting compartment controls

You can display an expansion/contraction control in the attributes and operations compartments of class and interface elements, and in the classes, interfaces and diagrams compartments of the package elements. Selecting the package, class or interface activates the control, which you can use to toggle the display of members in the compartment. This is particularly useful when you have large container elements with content that does not need to be visible at all times.

To set the compartment controls:

1. From the main menu, choose **Tools | Options | <level>**.
2. Select **Diagram** in the resulting Options dialog.
3. Check *Show controls for compartments*.

Sequence and collaboration diagrams

Sequence and collaboration diagrams model the dynamic aspects of a system. Both depict interactions described by a set of objects, their relationships, and the messages exchanged between them.

Collaboration diagrams emphasize the structural organization of objects, as illustrated by Figure 26. In contrast, sequence diagrams emphasize the time ordering of messages, as illustrated in Figure 27.

Collaboration diagrams are graphs; sequence diagrams are essentially tables with different objects and messages depicted across the *x*-axis and increasing time down the *y*-axis. However, the two diagrams are semantically equivalent: one type can convert to the other with no loss of information.

Although they are semantically equivalent, sequence and collaboration diagrams do not necessarily reveal the same information. For example, collaboration diagrams explicitly show how objects are linked, while in sequence diagrams the links are implied. Message return values display in sequence diagrams but not in collaboration diagrams.

Figure 26 Collaboration diagram

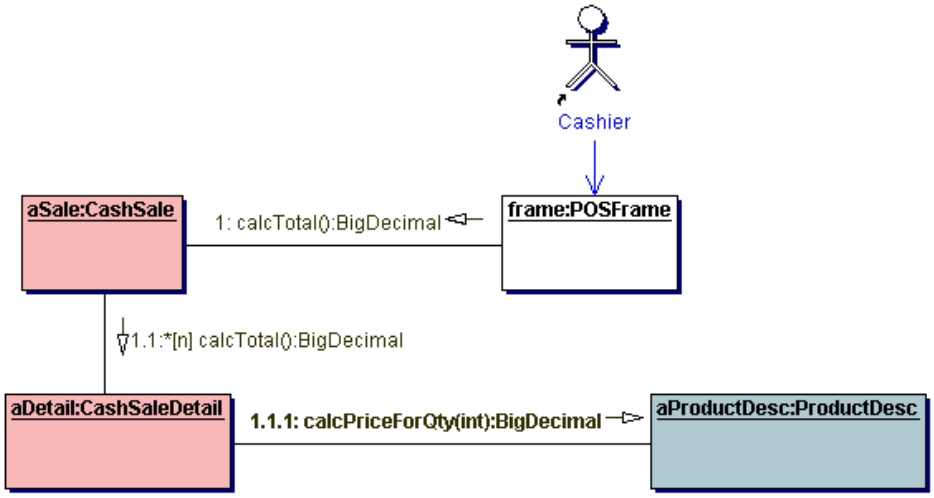


Figure 27 Sequence diagram

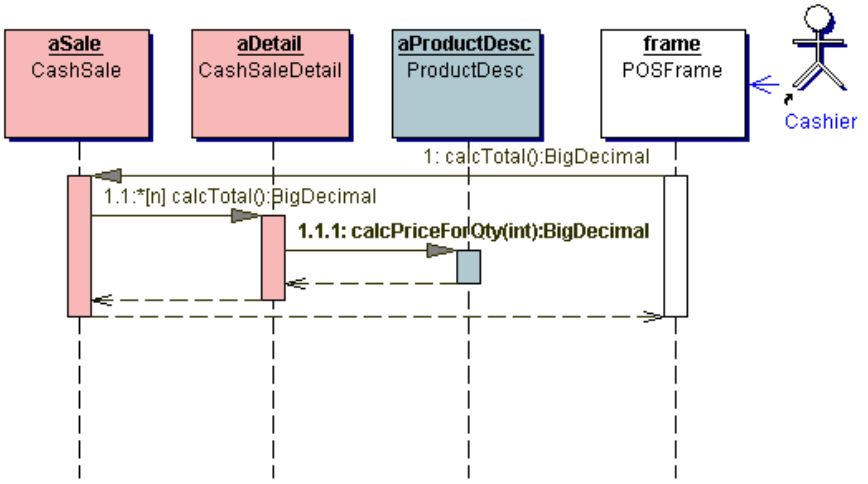


Diagram elements

Table 14 lists the elements of collaboration diagrams and Table 15 lists the elements of sequence diagrams. Relationship links are in *italicized* text.

Table 14 Elements of collaboration diagrams

Icon	Element
	Object
	Actor
	Message
	Self Message
	Associates
	Aggregates
	Note
	Note link

Table 15 Elements of sequence diagrams

Icon	Element
	Object
	Actor
	Message

Table 15 Elements of sequence diagrams (continued)

Icon	Element
	Message with delivery time
	Self Message
	Statement Block
	Note
	Note link

Converting between sequence and collaboration diagrams

You can convert between sequence and collaboration diagrams. However, when you create a new diagram, you must specify that it is either a sequence diagram or a collaboration diagram.

To convert between sequence and collaboration diagrams:

1. Right click on the diagram background.
2. If the diagram is a sequence diagram, choose **Show as Collaboration**. If viewing a collaboration diagram, choose **Show as Sequence.**
3. Repeat this process to switch back and forth.

Note You can also do this switch from the right-click menus of sequence and collaboration diagram nodes in the Explorer.

Creating sequence and collaboration diagrams

You can create sequence and collaboration diagrams and populate them using buttons on the diagram toolbar. You can also generate sequence diagrams from operations on a class diagram.

Note To generate a collaboration diagram from an operation, first generate the sequence diagram and then convert the diagram into a collaboration diagram.

Generating sequence diagrams from operations

To generate a sequence diagram from an operation:

1. Open the class diagram containing the class whose operation you want to model.
2. Locate the desired class and choose the desired operation.
3. Right click the operation and choose **Generate Sequence Diagram**. This opens the Generate Sequence Diagram, which lists packages and classes involved in the operation.

4. In the Package/class list, check the packages and classes you want to display in the generated diagram. All packages and classes are selected by default. However, some Java classes might not be relevant, such as `java.lang.Integer`. To increase the meaningfulness of the generated diagram, remove anything that is not helpful in explaining the sequence of operations.
5. For those elements you decide to show in the diagram, check whether or not to show implementation details in the generated diagram.
6. Click **Ok** to generate the diagram.

Together generates a new sequence diagram in the same package as the source class, with the same name as the source operation, and opens it in the Designer pane.

Setting options to control generation of sequence diagrams

The Options dialog enables you to set options for the generated diagrams.

To access sequence diagram options:

1. From the main menu, choose **Tools | Options | <level>**. This opens the Options dialog.
2. Expand **View Management | Sequence diagram**.

It is possible to generate sequence diagrams for several methods. You can display a sequence diagram for each method in a separate tab or show the sequence of calls for all selected methods in a single sequence diagram. To control this behavior, use the options *Create multiple diagrams* and *Show multiple diagrams.*

If *Create multiple diagrams* is checked, a separate diagram is generated for each selected method. Otherwise, sequence of calls for all methods is generated on a single diagram.

If *Show multiple diagrams* is checked, all diagrams open automatically after they are generated.

It is also possible to control nesting level. *Depth of call nesting* enables you to change nesting values according to the desired degrees of complexity.

Specifying classes for objects

You can specify a class for an object on a sequence or collaboration diagram.

To specify the class for an object:

1. Open the object's Inspector (Right click | Properties) and click on the **Properties** tab.
2. In the *instantiates* field, enter a value or click the browse button to pick the class from a dialog that lists all the resources available to the project.

The object displays its fully qualified name. The Inspector displays a Class tab in which you can access the properties of the object's class.

Tip You can also specify the classifier by right clicking the object and choosing Choose Class.

To create a shortcut to the object's class on the diagram, right click the object and choose Import Class.To remove its association with a class, right click the object and choose Unlink Class.

Working with messages

This section describes techniques for working with messages in sequence and collaboration diagrams. Although the two diagram types are equivalent, the techniques for dealing with messages differ.

Messages in collaboration diagrams

When you draw a message between objects, a generic link line displays between the objects and a list of messages is created above it. The link line is present as long as there is at least one message between the objects. Messages display in time-ordered sequence from top to bottom of the messages list. You can edit message properties in their Inspectors.

In addition to message links, you can add links to collaboration diagrams that show association and aggregation relationships. These links do not display if you view the diagram as a sequence diagram.

Messages in sequence diagrams

Messages in sequence diagrams have many more options than messages in collaboration diagrams. The options are on the Link tab of the message Inspector.

Creating a message-to-self

To create a message from an object back to itself:

1. Click the **Self Message** button on the diagram toolbar
2. For a sequence diagram, click the object's lifeline at the point where you want the message to appear.

Note To create a message to self in a collaboration diagram, click the Self Message button then click the object twice (for the start and the end of the message).

Creating a message link that corresponds to an operation call

To create a message link that corresponds to an operation call:

1. Create a message link between two objects. The recipient object must instantiate a class.
2. Open the Link tab of the message link Inspector.

3. Enter an operation in the *operation* field, or click the browse button to choose the operation.
4. Select the operation and click **Ok**. If the operation is not a member of the recipient's class, you must confirm whether to create a new operation or rename an existing one.

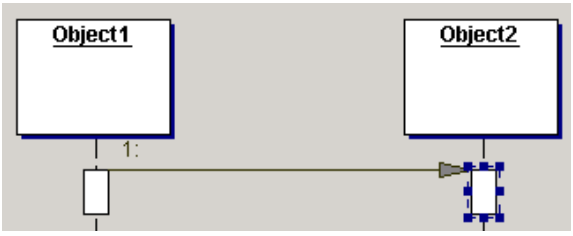
The message link is named according to the name of the operation. The Inspector for the link displays two tabs. The MessageLink tab displays message link properties and the Operation tab displays the operation's properties.

Tip You can create a new operation from a message by right clicking the message and choosing New | Operation or New | Constructor.

Changing activation times

Together automatically renders the activation of messages that show the period of time that the message is active. When you draw a message link to the destination object, the activation bar is created automatically, as shown on Figure 28.

Figure 28 Activation of an object on a sequence diagram.



You can extend or reduce the period of time of a message by vertically dragging the top or bottom line of the activation bar as required. A longer activation bar means a longer time period when the message is active.

Reordering message links

To reorder messages, drag message links up and down the object lifeline. Reordering automatically updates the message link numbers.

You can reorder message links keeping their sequential order and freeing the space between for new links. To do this, select a message link line, press CTRL, and drag it. This shifts all succeeding links. If you select several message links by pressing CTRL key, then those selected are moved keeping their increments.

Branching messages

You can branch together messages from the same activation bar using the Branching command of the message right-click menu. Right click the message and choose Branching | With previous.
To remove branching, right click the branched message and choose Branching | None.

Annotating message links

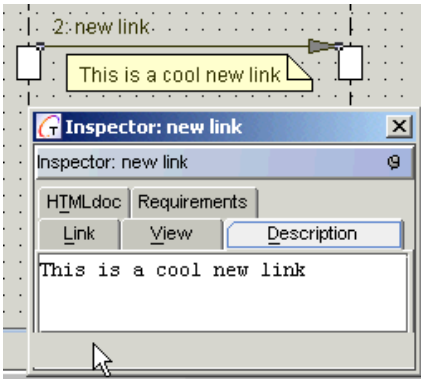
You can comment links using the Note button on the diagram toolbar. As the diagram changes, however, such notes do not move according to the messages they are attached to. To avoid this problem, annotate the links.

To enable link annotation:

1. On the main menu, choose **Tools | Options | <level>**.
2. Expand *View Management - Sequence diagram*.
3. Check *Show message description.*

To annotate a message link, use the Description tab of the message Inspector, as shown in Figure 29.

Figure 29 Message description

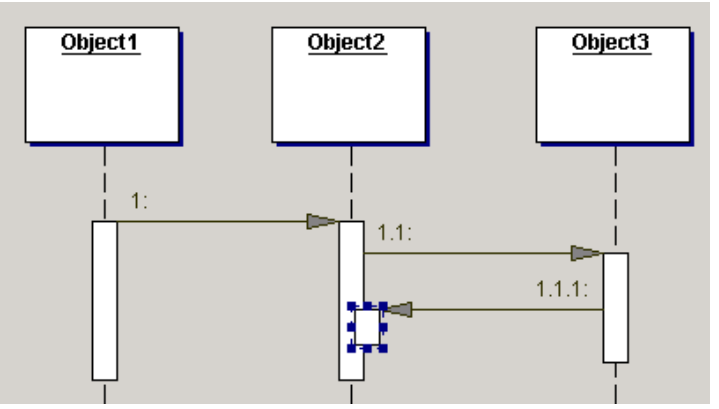


Nesting messages

You can nest messages by originating message links from an activation bar. The nested message inherits the numbering of the parent message. For example, if the parent message has the number 1, its first nested message is 1.1.

It is also possible to create message links back to the parent activation bars, as shown on Figure 34.

Figure 30 Nested messages



Creating statement blocks

When you place a statement block on an activation bar, a dialog opens for specifying which control structure the block represents. If the control structure requires a condition, you can enter the condition with the in-place editor.

Working with lifelines

Adjusting the default lifeline of objects

Objects display with a default lifeline when you place them on a sequence diagram. Their tops align vertically. If you draw a message to an object and then check the *creation* property of the message, the created object moves downward to show that it exists at a point later in time from its creator.

Adjusting the size of object lifelines

You can lengthen or shorten object lifelines as needed by dragging the horizontal line of the bottommost message link up or down. You can arrange the position of other intervening messages this way also.

Changing the order of objects in a sequence diagram

You can reorder objects in a sequence diagram and maintain any message links already created between them. Select any object and drag it across the *x*-axis of objects to the desired position. You cannot move objects vertically along the *y*-axis of time except as described in the first point above.

Marking a destroyed object with an "X"

Together can automatically render a bold **X** indicating the destruction of a created object. To use this feature, draw a message to the object and check the message's destruction property in the Link tab of the Inspector.

Generating implementation source code

You can generate a sequence diagram from source code. You can also generate source code from messages or activation bars of sequence diagrams. Each message in the call sequence must correspond to an actual operation.

To generate source code for a message or activation bar:

1. Right click the message or activation bar.
2. Choose **Generate Implementation**.

When the process is completed, labels of messages for which implementation code has been generated display in **bold** style on the diagram.

By default, Together does not generate code in any operations that do not have empty bodies. You can change this default behavior through the sequence diagram options.

Statechart diagrams

Statechart diagrams enable you to model the behaviors of objects as they goes through different states in response to events. Statechart diagrams show the possible states of an object and the transitions between states.

Figure 31 is an example of a statechart diagram.

Figure 31 Statechart diagram

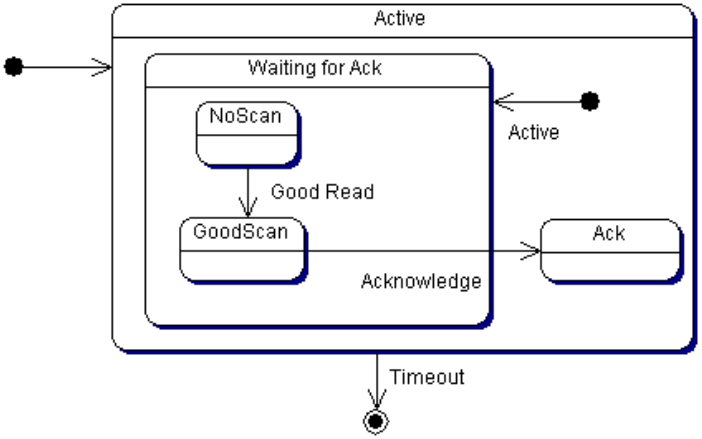


Diagram elements

Table 16 lists the elements of statechart diagrams that are on the diagram toolbar. Relationship links are in *italicized* text.

Table 16 Elements of statechart diagrams

Icon	Element
	State
	Start
	End
	History
	Horizontal Fork/Join
	Vertical Fork/Join
	Transition
	Object
	Note
	Note link

Drawing a self-transition

A self-transition flow leaves the state, dispatching any exit action(s), then re-enters the state, dispatching any entry action(s).

To create a self-transition:

1. Drag a transition link from the desired state and drop it on the diagram background.
2. In the resulting dialog, select the state where you began the link and click **Ok**.

Tip An alternative way to draw a self-transition is to draw a transition between two states, and then drag the opposite end of the link back to the desired state.

Creating internal transitions

An internal transition is a shorthand for handling events without leaving a state and dispatching its exit/entry actions.

To create an internal transition:

1. Select the desired state on the diagram
2. From the state's right-click menu choose **New Internal Transition**.

You can edit the event name in place. For additional information, see entry/exit actions below.

Specifying entry/exit actions for a state

Entry and exit actions are executed upon entering or leaving a state. You create entry and exit actions in Together statechart diagrams as stereotyped internal transitions.

To create an entry or exit action:

1. Create an internal transition in the desired state.
2. Edit the name of the transition in place, using the following syntax:
   `stereotype/actionName[argument]`

   For example: `exit/setState(idle)`

Optionally, you can create the internal transition and set the event name, event arguments, and action expression properties in the Inspector for the transition.

Creating nested substates

You can create a composite state by nesting one or more levels of states (i.e. substates) within one state. You can also place start/end states and history states inside a state and draw transitions between the contained substates. The easiest way to create a nested substate is to place a state element on the diagram background, then drag and drop it on top of another state.

Showing multiple transition sources or targets

A transition may have multiple sources, meaning it is a join from several concurrent states, or it may have multiple targets, meaning it is a fork to several concurrent states.

You can show multiple transitions in statechart diagrams with either a vertical or horizontal orientation. The statechart diagram toolbar provides separate fork/join buttons each one. The two orientations are semantically identical.

To create multiple transitions:

1. Identify the states involved. If necessary, place all the states on the diagram first and lay them out as desired.
2. Place a horizontal or vertical fork/join on the diagram and resize it as needed.
3. If depicting multiple sources, draw transitions from each of the source states to the fork/join, then draw a transition to the target state.
4. If depicting multiple targets, draw a transition from the source state to the fork/join; then draw transitions from the fork/join to each of the target states.

Modeling complex states

The techniques listed here pertain to models of complex composite states and substates.

- **Resize the main state.** You can resize the main state. You can essentially draw another statechart diagram inside it, complete with start/end/history states and transitions of all kinds to create a substate diagram.
- **Use hyperlinks to indicate nested substates.** You can nest multiple levels of substates inside one state. For especially complex substate modeling, however, it may be more convenient to create different diagrams, model each of the substate levels individually, and hyperlink the diagrams sequentially.
- **Apply existing content to a new diagram.** You can create a new diagram with the same content as the existing one by using the Clone command on the right-click menu of a diagram node in the Explorer. The new diagram has a unique name and is created in the same package as the original.
- **Reuse elements of existing diagrams.** You can reuse existing elements in other statechart diagrams by using the Add Shortcut command on the diagram's right-click menu. This opens the Add Shortcut dialog, in which you navigate to the existing statechart diagram and select elements, states, histories, and/or fork/joins.

Activity diagrams

An activity diagram is a flowchart that describes the flow of control from one activity to the next. You can show sequential and/or concurrent steps of a process, model business workflows, and model the flow control of an operation or the flow of an object as it passes though different states at different points in a process.

Figure 32 is an example of an activity diagram.

Diagram elements

Table 17 lists the elements of activity diagrams that are on the diagram toolbar. Relationship links are in *italicized* text.

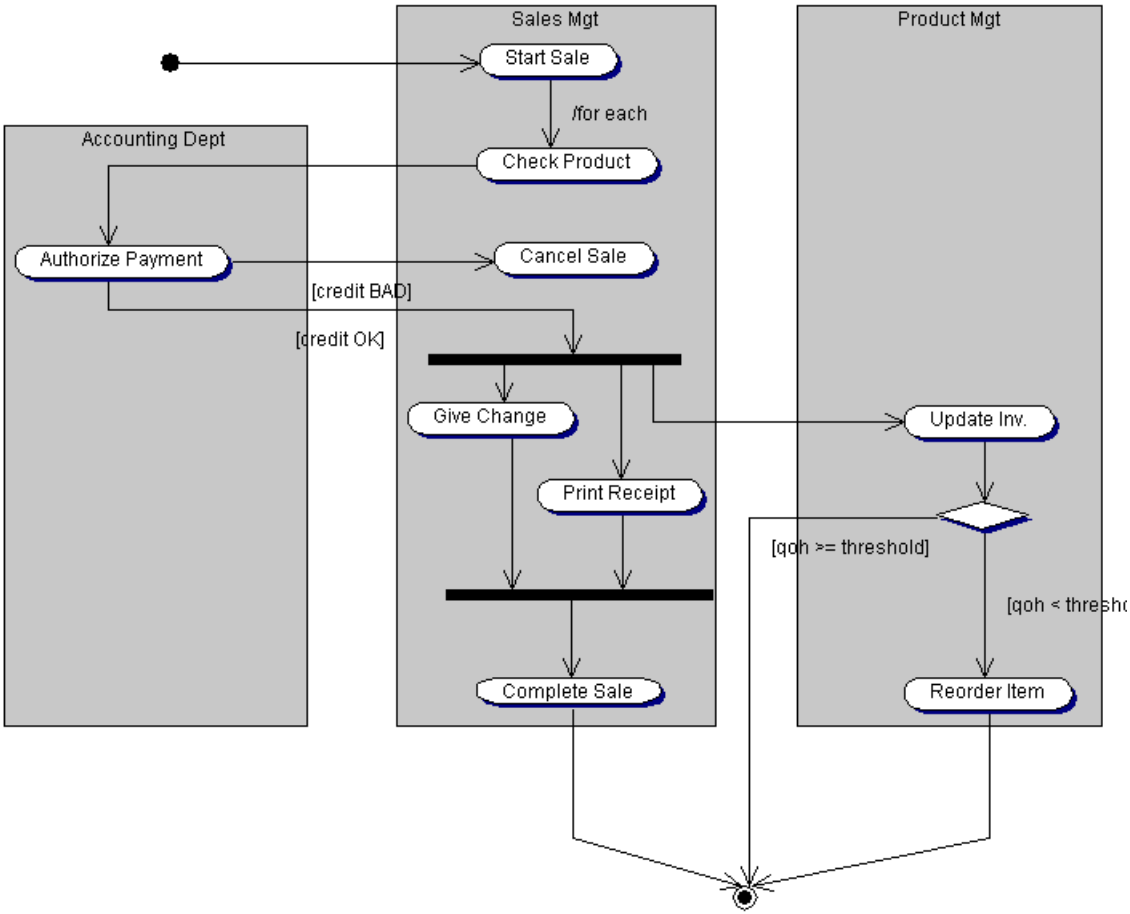
Table 17 Elements of activity diagrams

Icon	Element
	Activity
	Decision
	Signal Receipt
	Signal Sending

Table 17 Elements of activity diagrams (continued)

Icon	Element
	State
	History
	Object
	Start
	End
	Horizontal Fork/Join
	Vertical Fork/Join
	Swimlane
	Transition
	Object Flow
	Note
	Note link

Figure 32 Activity diagram



Working with activity diagrams

Following are tips and techniques that you can use when creating activity diagrams:

- **Create diagrams with identical content.** Non-trivial systems can require many activity diagrams to capture the dynamics of a workflow or operation. Use the Clone feature to create new diagrams with identical content in the same package.
- **Reuse elements of existing diagrams.** Invoke the Add Shortcut command on the diagram's right-click menu to reuse existing elements in other activity diagrams.

Note Elements imported this way are independent copies of the existing ones.

Start with the main flow modeling. Next, cover branching, concurrent flows, and object flows. Use separate diagrams as needed and hyperlink them for easy browsing later on.

- **Showing labels inside decisions.** To show labels inside decisions, go to the `$TGH\config\diagram.config` file and change the value of the option `option.show_decision_label` to true (it is set to false by default).

  Note that long labels may affect the diagram layout.

Component diagrams

Component diagrams show the components and interfaces of systems and the relationships between them. Figure 33 is an example of a component diagram.

A component is a container of other logical elements, and represents things that participate in the execution of a system. Components also use the services of other components via one of its interfaces. Components are typically used to visualize logical packages of source code (work product components), binary code (deployment components) or executable files (executions components).

Diagram elements

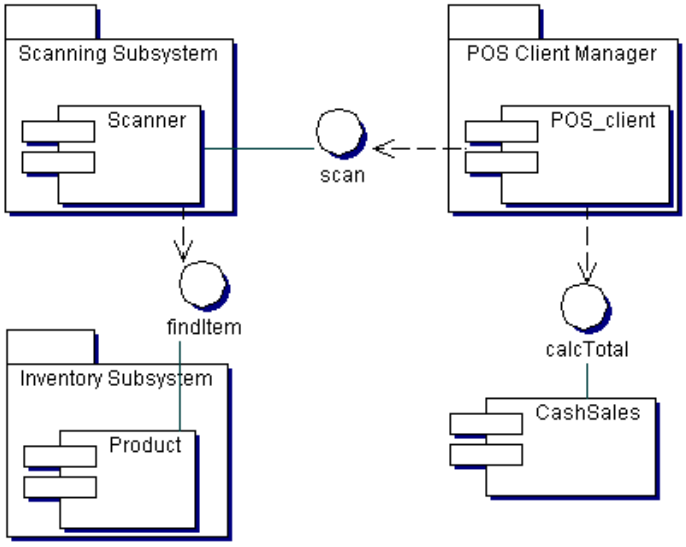
Component diagrams depict two kinds of relationships: dependency and realization. Component diagrams also can contain packages or subsystems that group logical elements of the model.

Table 18 lists the elements of component diagrams that are on the diagram toolbars. Relationship links are in *italicized* text.

Table 18 Elements of component diagrams

Icon	Element
	Subsystem
	Component
	Interface
	Supports
	Dependency
	Note
	Note link

Figure 33 Component diagram



Working with component diagrams

Following are tips and techniques that you can use when working with component diagrams:

- **Apply existing content to a new diagram.** Using the Clone command on the right-click menu of the navigation pane node, you can quickly create a new diagram with the same content as the existing one. The new diagram has a unique name and is created in the same package.
- **Reuse elements of existing diagrams.** Invoke the Add Shortcut command on the diagram's right-click menu to reuse existing elements in other component diagrams.

Note Elements imported this way are independent copies of the existing ones.

- **Represent realization.** You can represent realization in two ways: use the support stereotype and the dashed line (canonical form), and use the solid line (lollipop notation).
- **Hide subcomponents.** You can hide subcomponents on a component diagram. Choose Tools | Options | Default Level | View Management, and enter the following for one of the User Defined fields under Show:

```
Name: "Nested components"
Expression:
hasPropertyValue("$shapeType", "Component" ) && !(getContainingNode() ==
null)
```

Deployment diagrams

A deployment diagram provides a way to model the physical aspects of a computer system. It is a graph of nodes connected by communication associations, showing the physical architecture of the hardware and software of the system.

Figure 34 is an example of a deployment diagram.

Figure 34 Deployment diagram

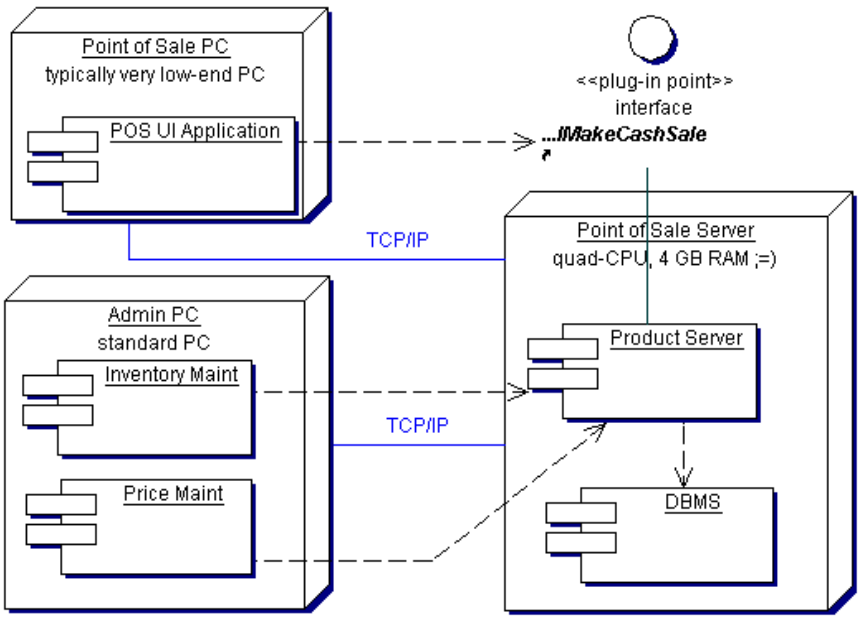


Diagram elements

Table 19 lists the elements of deployment diagram that are on the diagram toolbar. Relationship links are in *italicized* text.

Table 19 Elements of deployment diagrams

Icon	Element name
	Node
	Component
	Interface
	Supports

Table 19 Elements of deployment diagrams (continued) (continued)

Icon	Element name
	Associates
	Aggregates
	Object
	Dependency
	Note
	Note link

Working with deployment diagrams

Following are tips and techniques that you can use when working with deployment diagrams:

- **Apply existing content to a new diagram.** Using the Clone command on the right-click menu of the diagram node in the Explorer pane, you can create a new diagram with the same content as the existing one. The new diagram has a unique name and is created in the same package.
- **Reusing elements of existing diagrams.** Invoke the Add Shortcut command on the diagram's right-click menu to reuse existing elements in other deployment diagrams.

Note Elements imported this way are independent copies of the existing ones. Organize components by specifying the relationships between them.

- **Indicate a temporary relationship between a component and node.** Objects and components can migrate from one component instance to another component instance, and respectively from one node instance to another node instance. In such a case, the object (component) will be on its component (node) for only temporarily. To indicate this, use the dependency relationship with a *becomes* stereotype.
- **Represent how a component resides on a node.** A component may reside on nodes. You can represent this in two ways: use the *support* stereotype and the dashed arrows, or graphically nest the component element within the node element.

Chapter

10

UML Extension Diagrams

In addition to the UML diagrams discussed in Chapter 9, Together provides extension diagrams that are not currently supported by the UML standard. You can create these diagrams by choosing types from the Together tab of the New Diagram dialog. For more information, see "Creating diagrams in projects" on page 107.

This chapter includes the following topics:

- "Business process diagrams" on page 167.
- "Robustness analysis diagrams" on page 168..

Business process diagrams

Business process diagrams enable you to apply UML extensions to business modeling. Business object models depict the structure, processes, use cases, and relationships of a business. The business object model describes the realization of business use cases, providing an abstraction of how business workers and business entities are related and how they must work together to run the business.

A business object model describes the use cases of a business from the internal viewpoint of business workers. It defines the static and dynamic aspects of relationships between the workers and the classes and objects they use to produce the expected results. In aggregate, the objects of the model's classes are capable of performing all the use cases of the business.

Business process diagrams present the static aspects of a business object model. Sequence and activity diagrams can present the corresponding dynamic aspects.

Diagram elements

Table 20 lists the elements on the business process diagram toolbar. Relationship links are in *italicized* text.

Table 20 Elements of business process diagrams

Icon	Element
	Actor
	Use Case
	Business Worker
	Business Entity
	Communicates
	Extends
	Includes
	Generalization
	Aggregates
	Subscribes
	Associates
	System boundary
	Note
	Note link

Notation

The current UML specification does not determine graphical variations for elements such as use case and actor when used in a business object modeling context. Together business process diagrams use the standard UML graphical notation for these elements, and provides compliant notation for UML extensions such as business worker and business entity.

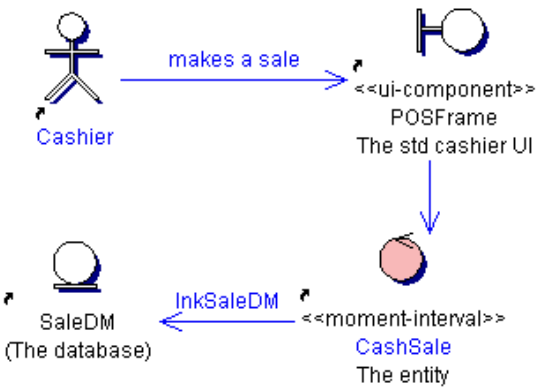
Robustness analysis diagrams

Robustness analysis involves evaluating the text of a use case and planning how to implement it using the objects you have discovered up to this point. A robustness model provides a bridge between the “analysis level” view described in the text of the use case and the “detailed design” view shown on a sequence diagram.

One of the main purposes of robustness analysis is to determine whether you have all the objects you need, and then add them to your class diagram. Robustness analysis suggests classifying objects into one of three stereotypes: boundary, control, and entity.

Figure 35 is an example of a robustness analysis diagram.

Figure 35 Robustness analysis diagram



It is beyond the scope of this document to provide detailed review of the robustness analysis. The most comprehensive information on this issue can be found in *Use Case Driven Object Modeling with UML* by Doug Rosenberg with Kendall Scott.

Diagram elements

Table 21 lists the elements on the robustness analysis diagram toolbar. Relationship links are in *italicized* text.

Table 21 Elements of robustness analysis diagrams

Icon	Element
	Boundary
	Entity
	Controller
	Worker boundary
	Worker controller
	Actor
	Association
	Robustness Association

Table 21 Elements of robustness analysis diagrams (continued)

	Note
	Note link

Key elements and properties

Robustness diagram elements such as entity, boundary, and worker boundary represent stereotyped views of classes. Each such element on the robustness diagram should correspond to a class. Controller and worker controller correspond to methods. Rosenberg's suggestion of using "invokes" in place of "includes" and "precedes" in place of "extends" is realized via robustness associations.

Key elements and properties of robustness diagrams include:

- **Boundary.** Objects in the new system with which the actors will be interacting. These may be windows, screens, dialogs, and menus.
- **Entity.** Database tables and files that store data, fetch data and perform computations that do not change frequently. These objects should be simple and generic enough to provide the possibility of future reuse.
- **Controller.** Objects that represent the functionality and system behavior of the use cases. Controllers may be converted into methods associated with the interface objects and entity objects.
- **Association.** Logical, directed associations.

  Associations follow these rules:

  - Actors can talk only to boundary objects.
  - Boundary objects can talk only to controllers and actors.
  - Entity objects can talk only to controllers.
  - Controllers can talk to both boundary objects and controllers but not the actors.

Part

4

GENERATING DOCUMENTATION

- Chapter 11, “Generating Documentation for Together Projects”
- Chapter 12, “Designing Custom Documentation Templates”
- Chapter 13, “Documentation Template Controls”
- Chapter 14, “Creating Multi-Frame HTML Documentation”
- Chapter 15, “Documentation Generator and Template Designer Reference”

Chapter

11

Generating Documentation for Together Projects

Together Edition for JBuilder can create external documentation for Together projects. This chapter describes how to use the documentation generator in Together to create project reports from within Together and from the command line.

The contents of this chapter are:

- "Generating documentation for an open project" on page 173.
- "Automating documentation generation" on page 182.

Generating documentation for an open project

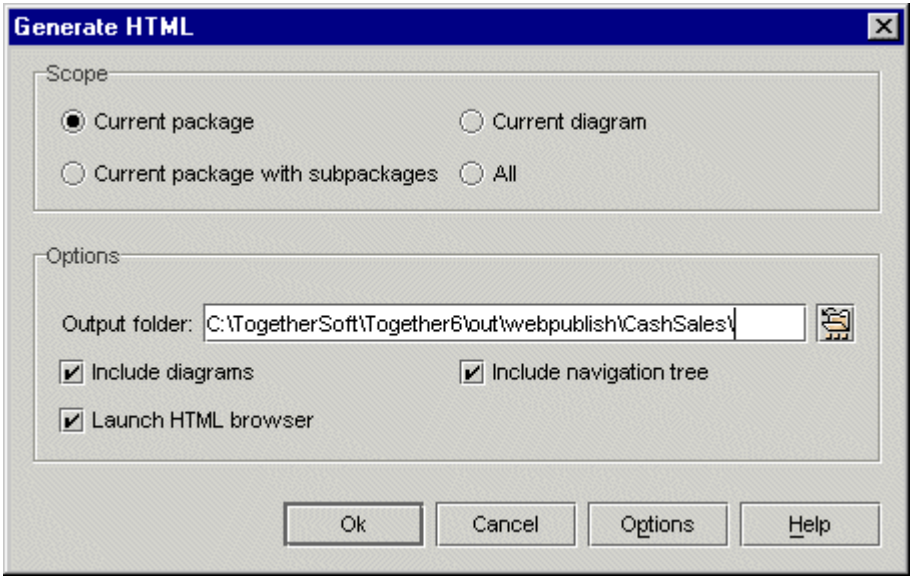
You can start the documentation generator from an open Together project. Simply select one of the following three commands from the main menu in Together:

1. **Project | Documentation | Generate HTML**: generates HTML documentation using a documentation template that Together provides.
2. **File | Print Documentation**: generates PDF documentation or prints directly to a printer using a documentation template that Together provides.
3. **Project | Documentation | Generate Using Template**: generates HTML, RTF, or text documentation using a custom documentation template or a template that ships with Together.

Generating HTML documentation

The Generate HTML command generates HTML documentation using a documentation template that Together provides. Figure 36 shows the Generate HTML dialog box.

Figure 36 Generate HTML dialog box



Generate HTML documentation generation parameters

The dialog box for generating HTML documentation allows you to fill in the parameters that determine the resulting report. The *Scope* section at the top of the dialog box has radio buttons to indicate what parts of the project should be parsed and included in the generated output.

- *Current package:* Generated output includes only the current package selected in the Model tab of the Explorer.
- *Current package with subpackages:* Generated output includes the current package selected in the Model tab of the Explorer and any subpackages under it.
- *Current diagram:* Generated output for the current diagram that is in focus in the Designer pane.
- *All:* Generated output covers the entire project.

The lower section of the dialog box has options to specify the destination and optional actions.

- *Output folder:* Enter the location for the generated output, or select from the file chooser. The default location for HTML documentation is `$TGH$/out/webpublish/$PROJECT_NAME$`.

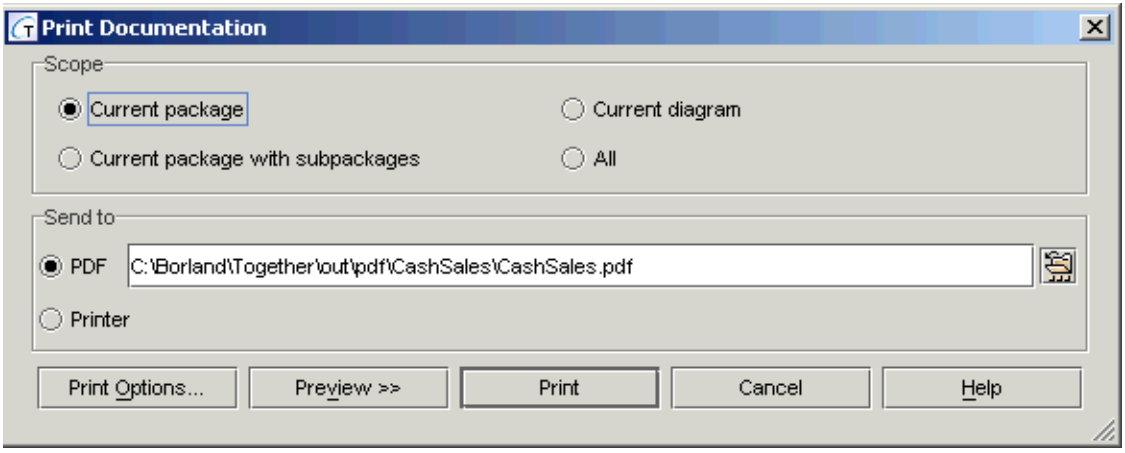
Important Each time that documentation is generated, any previously generated documentation files will be overwritten.

- Checkboxes:
  - *Include diagrams:* Check to include diagram images in the output .
  - *Include navigation tree:* Check to include a navigation tree in the output.
  - *Launch HTML browser:* Check to load the documentation into the default web browser of the operating system. This starts the application if necessary.
- *Options button:* Click the **Options** button to refine additional parameters for generating HTML documentation.

Printing documentation

The Print Documentation command generates PDF documentation or prints directly to a printer using a documentation template that Together provides. Figure 37 shows the Print Documentation dialog box.

Figure 37 Print Documentation dialog box



Print Documentation generation parameters

The dialog box for printing documentation in PDF format or to a printer enables you to fill in the parameters that determine the resulting report. The *Scope* section at the top of the dialog box has radio buttons to indicate what parts of the project should be parsed and included in the generated output.

- *Current package:* Generated output includes only the current package selected in the Model tab of the Explorer.
- *Current package with subpackages:* Generated output includes the current package selected in the Model tab of the Explorer and any subpackages under it.

- *Current diagram:* Generated output for the current diagram that is in focus in the Designer pane.
- *All:* Generated output covers the entire project.

The lower section of the dialog allows you to specify the destination for PDF documentation. Specify printer options and display a preview of the documentation prior to printing using the **Print Options** and **Preview** buttons at the bottom of the dialog.

- *PDF:* Enter the location for the generated PDF file, or select from the file chooser. The default location for PDF documentation is `$TGH$/out/pdf/$PROJECT_NAME$`.

Important Each time that documentation is generated, any previously generated documentation files will be overwritten.

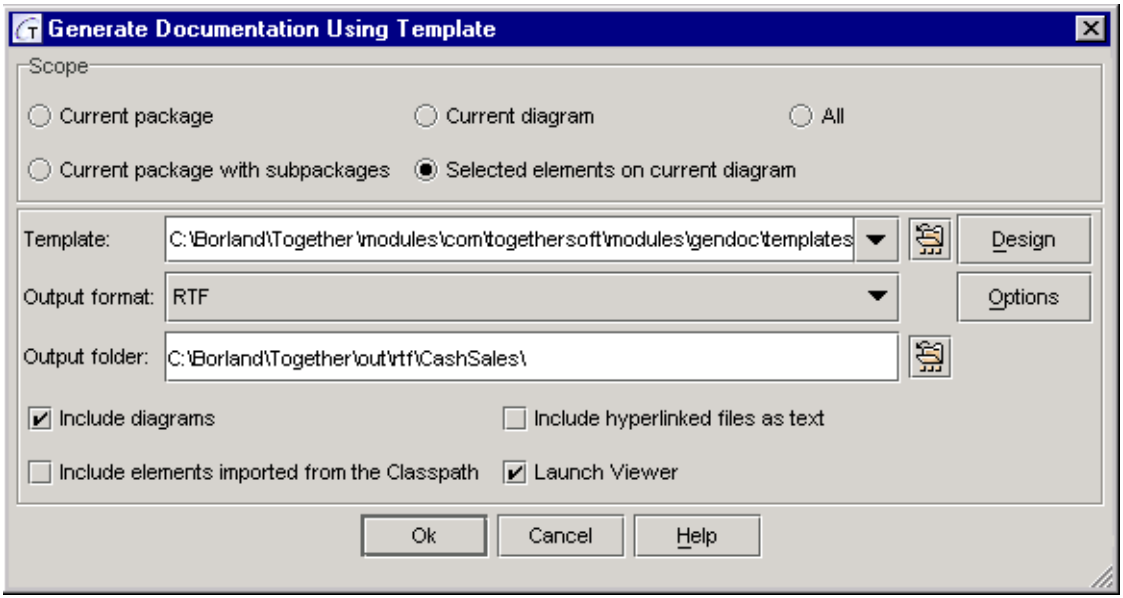
- *Printer*: Select to print documentation to a printer.
- Buttons:
  - *Print Options:* Click the **Print Options** button to refine printer parameters.
  - *Preview:* Click the **Preview** button to preview the documentation.

Generating documentation using a template

The *Documentation Generator*, or *DocGen*, is the Together module that produces project reports. DocGen creates project reports by applying *documentation templates* to Together projects. The templates contain commands to the DocGen engine; the projects provide the source of project-specific data. Documentation templates are `*.tpl` text files with formatting instructions and tags for the commands.

The Generate Using Template command generates HTML, RTF, or text documentation using a custom documentation template or a template that ships with Together. Figure 38 shows the Document Generation Using Template dialog box.

Figure 38 Document Generation dialog box



Documentation generation parameters

The dialog box for generating documentation from a template enables you to fill in the parameters that determine the resulting report. The *Scope* section at the top of the dialog box has radio buttons to indicate what parts of the project should be parsed and included in the generated output.

- *Current package:* Generated output includes only the current package selected in the Model tab of the Explorer.
- *Current package with subpackages:* Generated output includes the current package selected in the Model tab of the Explorer and any subpackages under it.
- *Current diagram:* Generated output for the current diagram that is in focus in the Designer pane.
- *Selected elements on current diagram:* Generated output only for the elements that are selected in the current diagram.
- *All:* Generated output covers the entire project.

The lower section of the dialog box has options to specify the type of template from which the documentation is to be generated, output format and destination, and optional actions.

- *Template:* Specify the template for the documentation generator to use. Select from the history dropdown list or press the file chooser button to select one of the available templates. By default, the file chooser navigates to the `templates` folder of the `gendoc` module.

- *Design:* Click to create your own template or modify an existing one. (Extensive information on how to design custom templates starts on Chapter 12, "Designing Custom Documentation Templates.")
- *Output format:* Select HTML, RTF, or TXT from the dropdown list.
- *Output folder:* Enter the location for the generated output, or select from the file chooser. The default location is a subfolder of `$TGH$/out`. The subfolder corresponds to the *Output format* selected. For example, if you choose HTML as the *Output format*, then the generated output location of the documentation will be `$TGH$/out/html/$PROJECT_NAME$`.

Important Each time that documentation is generated, any previously generated documentation files will be overwritten.

- Checkboxes:
  - *Include diagrams:* Check to include diagram images in the output (according to the instructions in the template).
  - *Include elements imported from the classpath:* Check to expand the generated documentation to include the classpath elements.
  - *Include hyperlinked files as text:* Leave this box unchecked to embed all hyperlinks to external documentation as URLs. Check the box to embed the hyperlinked documentation in the generated document. With RTF documentation, the embedded documentation is as follows:

    INCLUDE TEXT:<URL> To update the field, select it and press F9 or press Ctrl-A, F9 to update all fields. Mac OS X users should press Open Apple-A, F9.
  - *Launch viewer:* Check to load the documentation into the appropriate application as soon as the documentation is generated. This starts the application if necessary.

Note HTML documentation launches in the default web browser of the Operating System. RTF documentation launches in the default RTF viewer of the Operating System. Text documentation launches in the JBuilder Editor.

RTF documentation options

With RTF as the output format, you can click the **Options** button to refine the parameters for generating the documentation. The available options (shown in Figure 39) are as follows:

- *Formatting template:* Enter the fully-qualified name of a formatting template (such as a Microsoft Word `*.dot` file), or pick a template with the file chooser button.
- *Render HTML tags:* Check to translate HTML tags into appropriately formatted text in the printed documentation. The following tags are supported:

`<b>,<i>,<u>,<h1>,...,<h6>, <code>, <tt>, <em>, <font color>, <pre>, <p>, <br>, <ol>, <ul>, <li>`

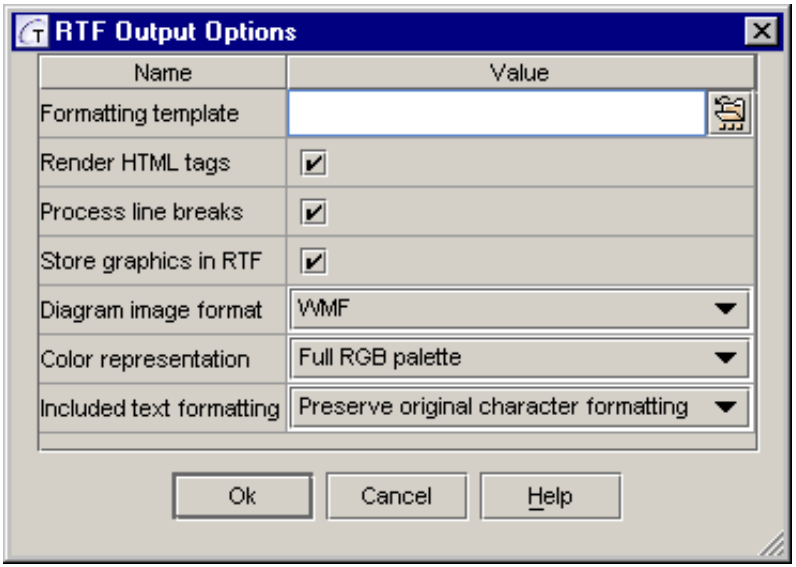
Note Descriptions for model elements may contain HTML tags. This setting is useful for documentation templates that process descriptions.

- *Process line breaks:* Check to preserve line breaks even if the Render HTML tags is checked on.
- *Store graphics in RTF:* Check to embed all the graphics in a single RTF document.

Note RTF supports PNG rather than GIF image format. Storing GIF images may cause problems with some RTF viewers. Since Microsoft Word 2000 handles GIF images by translating them to PNG images, you can save the generated documentation in Word in order to make that translation. Be aware, however, that you may have to make minor changes to the file (such as remove a space) in order to format the document correctly.

- *Diagram image format:* Select WMF or GIF format.
- *Color representation:* Select RGB color or 16-bit color.

Figure 39 RTF Output format options



Using a custom formatting template (.dot file) with DocGen

To use a custom formatting template (`.dot` file), the DocGen template must be modified to contain the same set of styles that are defined in the `.dot` file. If the set of styles contained in the `.dot` are not defined in the DocGen template, those styles will not be used by DocGen when generating documentation.

To use a custom formatting template (`.dot` file) with DocGen:

1. Specify the path to the `.dot` file in the RTF documentation options. (For more information, see "RTF documentation options" on page 178.)
2. Open the Documentation Designer. From the main menu, choose **Project | Documentation | Design Templates.**
3. In the Documentation Template Designer, choose **File | Properties**.
4. Choose the **Formatting Styles** tab.
5. Click the **New** button.
6. Choose the **Main** tab.
7. In the **Name** textbox, specify the style name defined in the `.dot` file.

To apply the style, it must be associated with an area property or control in the DocGen template. The style for each area property or control must be set separately.

Note Area property style settings override style settings that have been made to controls within the same area.

To set the Style for an Area Property:

1. Right click within a **Static Section** on the template.
2. Choose **Area Properties** from the right-click menu.
3. Choose the **Other** tab.
4. Choose the appropriate **Formatting Style** from the drop-down list.
5. Click **Ok**.

For more information on area properties and controls, see "Setting area properties" and "Creating controls and setting control properties" on page 205.

Generating reports with the Rational® RequisitePro® plug-in

The RequisitePro integration allows you to generate reports for your Together project. You can use DocGen to generate the reports in HTML, RTF, or text.

Setting up the reporting tool

The RequisitePro integration enhances DocGen, and requires some additional setup before you can use the reporting tool. Use a text editor to complete the instructions in this section.

To set up the reporting tool for use with RequisitePro, you need the following files:

- `$TGH$\modules\com\togethersoft\modules\gendoc\templates\MetaModel.mm`
- `$TGH$\ modules\com\togethersoft\modules\reqpro\ReqProMetaModel.mm`

Using a text editor, add the contents of the ReqProMetaModel file to the end of the MetaModel file.

Running reports

The instructions in this section assume that you have completed the steps to set up the reporting tool in the previous section. To run reports for RequisitePro, the document generation tool in Together uses the template, report.tpl, as described in the instructions that follow. Note that you can also modify this template for your specific needs.

To generate a report using the standard template:

1. From the main menu, choose **Project | Documentation | Generate Using Template**.
2. Browse to select the template file report.tpl located in the `$TGH$\modules\com\togethersoft\modules\reqpro\report.tpl` directory.
3. Click **Ok**.

Your browser should appear with a report. For more information on report generation, see , "Generating documentation using a template" on page 176.

To modify the existing template:

1. Save the default template as a backup.
2. Choose **Project | Documentation | Design Templates**.
3. Open the template file.

Refer to Chapter 12, "Designing Custom Documentation Templates" on page 187 for more information on designing custom documentation report templates.

Documentation Generation Tips

This section provides tips that may be used when generating documentation for your projects.

Adding package-level documentation

When generating HTML documentation in Together (**Project | Documentation | Generate HTML**), documentation for any diagram can be added by updating the description property for the diagram. However, you can also document the physical package in your project by creating a separate file named, `package.html`. Place the `package.html` file into the appropriate source code directory, and it is read during the documentation generation process.

To add package level documentation, perform the following steps:

1. Create the `package.html` file. An example is shown below:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">

<html>

<body bgcolor="white">

<h2>Package Specification</h2>
```

```
<ul>
  <li><a href="">#  Refer to any framemaker specification here #</a>
</ul>


<h2>Related Documentation</h2>


For overviews, tutorials, examples, guides, and tool documentation,
please see:
<ul>
  <li><a href="">#  Refer to non-spec documentation here #</a>
</ul>
</body>
</html>
```

The documentation generator processes `package.html`, and copies everything between the `<body>` tags to the destination file, `package-summary.html`.

2. Save `package.html` to the appropriate source code directory.
3. Generate HTML documentation through Together. From the main menu select, **Project | Documentation | Generate HTML**.

Line breaks in PDF documentation

By default, Together preserves line breaks in the PDF documentation (**File | Print Documentation**). To disable this, edit `$TGH$\config\print.config` as follows:

```
print.doc.process.line.breaks = false
```

Automating documentation generation

You can update project documentation as part of a periodic automated build process by having the process script call the documentation generator in Together through the command line interface. To generate documentation from the command line without launching Together, use the following launchers:

- `$TGH$/bin/TogetherCon.exe` (Windows)
- `$TGH$/bin/Together.bat` (Windows)
- `$TGH$/bin/win32/umldoc.exe` (Windows)

Both `TogetherCon.exe` and `Together.bat` produce RTF and HTML documentation using modifiable template files (`*.tpl` files) as well as regular HTML documentation. `umldoc.exe` produces HTML documentation only. All of the

launchers bypass the Together window. Table 22 lists each of the launchers to be used for automating documentation generation and the type of documentation that the launcher generates.

Table 22 Launchers and documentation type

Launcher	Documentation Type
`TogetherCon.exe` `Together.bat`	• Documentation generated as RTF, HTML, or text using a modifiable template such as `ProjectReport.tpl` • Documentation generated as regular HTML
`umldoc.exe`	Documentation generated as regular HTML

Launching the documentation generator from the command line

To generate RTF or HTML documentation from the command line without launching Together use the following command:

```
<launcher> [module] [options] <project>
```

where:

- `<launcher>` is Together`Con.exe`, Together`.bat`, or `umldoc.exe`.
- `[module]` is a command for the specific utility:

  `-script=com.togethersoft.modules.genhtml.GenerateHTML` for generating HTML documentation. Using this module is analogous to using **Project | Documentation | Generate HTML** from within Together.

  `-script=com.togethersoft.modules.gendoc.GenerateDocumentation` for generating RTF or HTML documentation. Using this module is analogous to using **Project | Documentation | Generate Using Template** from within Together.

Note The `script` parameter is not required when using `umldoc.exe` since it executes the `GenerateHTML` module only.

- `[options]` are those for the `GenerateDocumentation` or `GenerateHTML` modules. Options are listed in Table 23 and Table 24.
- `<project>` is the absolute path to the project, for example: `C:`/Borland/Together`/samples/java/CashSales/CashSales.tpr`

At a bare minimum, the command needed to launch the `GenerateDocumentation` or `GenerateHTML` modules using Together`Con.exe` or Together`.bat` is:

```
<launcher> [module] <project> -d <output directory>
```

Tip Using the `GenerateDocumentation` module in the above command will generate RTF documentation by default. To change the output format to HTML or text, use the `format` option: `-format HTML` or `-format TXT`.

At a bare minimum, the command needed to launch the `GenerateHTML` module using the `umldoc.exe` launcher is:

```
<launcher > <project> -d <output directory>
```

Where, `output directory`, is the absolute path to the output directory.

Table 23 lists the available command line options for generating documentation using a template. Table 24 lists the available command line options for generating regular HTML documentation.

Note The Together`.exe` and Together`.sh` launchers do not presently run fully in console mode. The Together shell opens displaying the Generate Documentation dialog box for setting the documentation generation options.

Documentation generation options

Generating automatic HTML or RTF documentation from a template (for example, `ProjectReport.tpl`) or generating regular HTML documentation allows for specific command-line options. Use the options described in these sections when generating documentation from the command line.

Options for generating documentation using a template

The options described in this section pertain to documentation generated as RTF, HTML, or text by using a modifiable template file, such as `ProjectReport.tpl`. Generating this type of documentation is the same as generating it from within Together by using **Project | Documentation | Generate Using Template**.

Each documentation option is either a “switch option” or a “parameter option.” Switch options have the form:

-option_name

Parameter options are followed by parameter values:

-option_name parameter_value

There are regular documentation using template options and custom options. The regular options are listed in Table 23.

Custom options are those not recognized as regular options. There is no fixed list of custom options, and custom options are not processed directly by the Documentation Generator. Each custom option must have a parameter.

Custom options are stored with templates. You can access the name of any custom option within a template using this function call:

```
getDGOption(String optionName)
```

This feature allows you to pass parameters to a template from the command line and adjust the template behavior dynamically.

Table 23 Command line options for generating documentation using a template

Command	Description
-template <path>	The template file. If omitted, the default "ProjectReport.tpl" template is used: ..modules\gendoc\templates\ProjectReport.tpl
-metamodel <path>	The MetaModel file. If omitted, the default meta-model file is used: ..modules\gendoc\templates\MetaModel.mm
-format <RTF|HTML|TXT>	Output format; RTF is the default.
-styletemplate <path>	Style template file (depends on the output format such as *.dot file for RTF output)
-d <directory>	Output destination directory
-f <path>	Output file path, compatible with TXT output format only. This option redirects all output to the specified file.
-diagrams	Include diagram images.
-recurse	Create output for packages specified in [packagenames] and their subpackages.

Options for generating HTML documentation

The options described in this section pertain to regular HTML documentation. Generating this type of documentation from the command line is analogous to generating it from within Together by using **Project | Documentation | Generate HTML**.

The options in Table 24 are valid for the umldoc.exe launcher. These options also apply when using the GenerateHTML module with the Together.bat or TogetherCon.exe launchers.

Table 24 Command line options generating regular HTML documentation

Command	Description
-overview <file>	Read overview documentation from HTML file.
-public	Show only public classes and members.
-protected	Show protected and public classes and members (default).
-package	Show package/protected/public classes and members.
-private	Show all classes and members.
-help	Display command line options.
-sourcepath <pathlist>	Specify where to find source files.
-classpath <pathlist>	Specify where to find user class files.
-d <directory>	Destination directory for output files. ***Note:*** Not required for Together.exe, since this initiates the Together documentation generation dialog.

Table 24 Command line options generating regular HTML documentation (continued)

-use	Create class and package usage pages.
-version	Include @version paragraphs.
-author	Include @author paragraphs.
-splitindex	Split index into one file per letter.
-windowtitle <text>	Browser window title for the documentation.
-doctitle <html-code>	Include title for the package index (first) page.
-header <html-code>	Include header text for each page.
-footer <html-code>	Include footer text for each page.
-bottom <html-code>	Include bottom text for each page.
-nodeprecated	Do not include @deprecated information
-nodeprecatedlist	Do not generate deprecated list.
-notree	Do not generate class hierarchy.
-noindex	Do not generate index.
-nohelp	Do not generate help link.
-nonavbar	Do not generate navigation bar.
-stylesheetfile <path>	File to change style of the generated documentation.
-togethersettings	Use settings from TGH/config/GenerateHTML.config file.
-recurse	Create output for packages specified in [package names] and their subpackages.
-javadoc	Create the same output as javadoc.exe produces.
-browser	Launch HTML browser.
-diagrams	Include diagram images.
-navtree	Generate navigation tree.
-nopackagedependencies	Do not show package dependencies in all class diagrams.

Chapter

12

Designing Custom Documentation Templates

The *Documentation Generator*, or *DocGen*, is the Together module that produces project reports. DocGen creates project reports by applying *documentation templates* to Together projects. The templates contain commands to the DocGen engine; the projects provide the source of project-specific data. Documentation templates are `*.tpl` text files with formatting instructions and tags for the commands.

The *Documentation Template Designer* is a tool in Together for creating custom documentation templates. This chapter shows how to design documentation templates. The remainder of this part of the User Guide covers the details of documentation template creation.

The contents of this chapter are:

- "Template Organization" on page 187.
- "Metamodel types" on page 190.
- "Creating new templates" on page 192.
- "Designing the template body" on page 194.
- "Accessing model elements through iterator sections" on page 196.
- "Reusing templates and stock sections" on page 200.

Template Organization

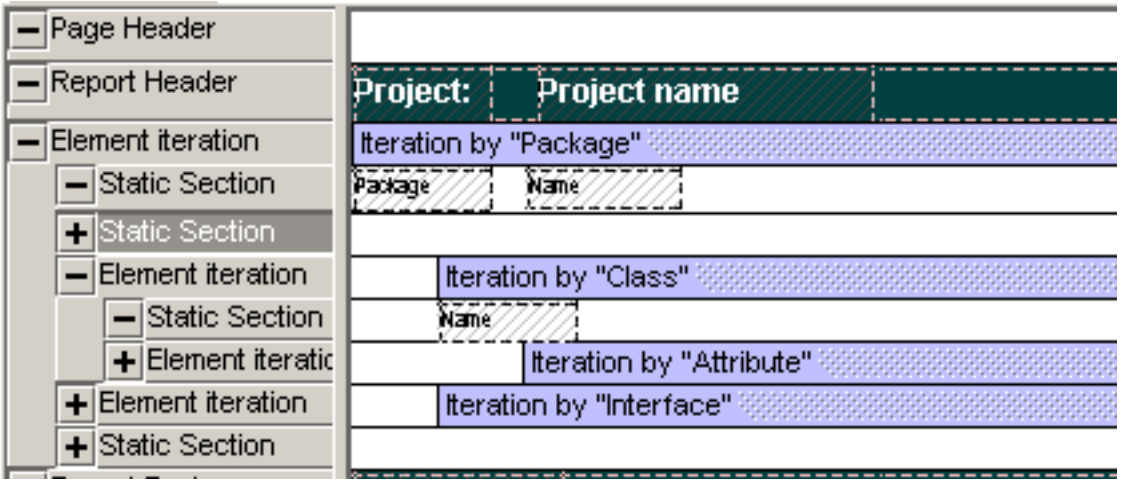
To open the Designer, select **Project | Documentation | Design Templates** from Together's main menu**.** Alternatively, click the **Design** button when generating documentation from a template.

The Documentation Template Designer divides templates into five major zones:

1. Page Header
2. Report Header
3. Body
4. Report Footer
5. Page Footer

The zones are horizontal bands that go across two panes. The *scope pane*, which is on the left, reveals the template structure. The *details pane* on the right shows the contents of the zones, which include commands to the DocGen engine. (See Figure 40.)

Figure 40 Documentation Template Designer



Headers and footers are at the top and the bottom of the Designer window. The report header and the report footer apply only once per document. Page headers and footers apply once per page for RTF documentation; they are ignored for HTML and text documentation.

The body zone of a template contains the commands that produce the body of the generated report. DocGen builds a report into horizontal regions. Each region in the report corresponds to a section in the template that determines the data for that region and how that data should appear.

Body sections and the DocGen engine

The body of a documentation template is organized into a hierarchy of sections. The body of the template in Figure 40 starts with the first Element iteration section and ends with the last Static Section.

Some sections in the body are nested inside others. Some sections have siblings. Thus, the sections form a tree whose root is the entire body zone. Sections that are not nested within any others are children of the root. The root of the tree in Figure 40 contains exactly one child (the first Element iteration section). The scope pane reveals the tree structure, indenting each section according to its level in the tree.

Every section in the body has a *section scope*. Scopes are based on *metatypes* that correspond to the different types of model elements. The section scope of the body zone corresponds to the *root object metatype.*

Note Model elements can be thought of as directories, files, any visible on-screen widgets, as well as their properties that show up in the property inspector. This includes diagrams, actors, packages, classes, attributes, swimlanes, and so on.

The model itself is considered to be a special metatype, which is the default root metatype when you create a new template.

The DocGen engine uses a dynamic *current model element* to go through a template and access specific project information. The type of the current element is the metatype for the section that the engine is currently processing. The value of the current element changes according to when the processing for the section takes place.

DocGen produces the body of a report starting from the root element, going in a "depth-first" fashion. In other words, DocGen starts its processing with the first root section, visiting it along with any of its nested subsections before continuing to the next root section. This pattern is recursive: visit the subtree rooted at a section before going to the next sibling section. For each sibling of a section, DocGen begins its processing with the same current element.

Body section types

There are six different types of body zone sections.

1. **Static sections** contain the commands to DocGen for getting project data. (You can find detailed information on commands in Chapter 13, "Documentation Template Controls.")
2. **Element iterators** are for looping through model elements of a particular type, applying commands in its nested sections.
3. **Element property iterators** are for looping through the properties of model elements.
4. **Folders** group other sections.
5. **Calls to stock sections** reuse sections from the template's internal collection of stock elements.
6. **Calls to template sections** use other templates.

Static sections and iteration sections provide most of the functionality required by most templates. Folders, calls to stock sections, and calls to templates are conveniences for organizing and reusing templates. Detailed information on creating body sections is covered later in "Creating new templates" on page 192.

Metamodel types

Documentation metamodels define the types that determine the section scopes. The metamodels are described in two model definition files, `MetaModel.mm` and `GUIBuilderModel.mdf`. These are text files in the Together installation under the folder `$TGH$\modules\com\togethersoft\modules\gendoc\templates`.

The model definition files show how shape types correspond to model elements, the types of elements another element can contain, and the properties of each element type

The beginning of each model definition file lists the properties that DocGen knows. These include *RWI* properties, which are defined in the read-write interface of Together's open API, DocGen properties, and others.

The remainder of each model definition file contains the metatype definitions. The major fields in the definitions are as follows:

- `name`: metatype name
- `extends`: parent metatype
- `rwi_entity`: the name of the related element in the RWI of Together's open API
- `full_name`: the name displayed in the Documentation Template Designer
- `properties`: a list of properties available for this type (DocGen properties, RWI properties, and others). Every metatype except `MODEL` has a `$shapetype` property.
- `excluded_properties`: items listed among the properties that are not available for this type (used for inheritance)
- `metatype_filter`: filtering conditions that distinguish among elements with the same shape type
- `contained_metatypes`: child metatypes that can be directly obtained from any element of this metatype

The name field for each type is always present. The existence of the other fields varies with the type. DocGen uses shape types, metatype filter expressions, RWI entity types, and collections of child metatypes to determine the relationships among the model elements.

Below is the code from `MetaModel.mm` that describes the `ELEMENT`, `DIAGRAM`, and `NODE` metatypes.

```
#-------------------------------------------------------------------
```

```
<metatype>
  name=ELEMENT
  rwi_entity=element
  full_name="[gendoc/gen_doc_by_template1/full_name_ELEMENT]"
  properties = { $shapeType; $name; $doc; %annotation;
                 $hyperlink; url }
  excluded_properties = { $shapeType; $name }
  contained_metatypes = { HYPERLINK_LINK }
</metatype>
#-------------------------------------------------------------------
<metatype>
  name=DIAGRAM
  extends=ELEMENT
  rwi_entity=diagram
  full_name="[gendoc/gen_doc_by_template1/full_name_DIAGRAM]"
  big_icon = "DiagramTypes/Default.gif"
  properties = { %package; stereotype; alias }
  contained_metatypes = { DIAGRAM_REFERENCE; DIAGRAM; NODE; LINK }
</metatype>
#-------------------------------------------------------------------
<metatype>
  name=NODE
  extends=ELEMENT
  rwi_entity=node
  full_name="[gendoc/gen_doc_by_template1/full_name_NODE]"
  properties = { %package }
  contained_metatypes = { NODE; MEMBER; LINK }
</metatype>
#-------------------------------------------------------------------
```

An element iterator or folder can contain nested element iterators whose type is listed among its contained metatypes, the contained metatypes of its parent, or indirectly through the contained metatypes of one of its contained metatypes. For example, an element iterator with `DIAGRAM` scope can contain nested element iterators with the following scopes:

- hyperlink (inherited from ELEMENT)
- diagram reference, diagram, node, link
- member (indirectly through the contained metatype, NODE)

Element properties are inherited. An element iterator can contain nested property iterators whose type is inherited from its ancestor or listed directly among its properties. For example, an element iterator with DIAGRAM scope can contain nested property iterators for the following type scopes:

- shapetype, name, documentation, annotation, hyperlink, url (inherited from ELEMENT)
- package, stereotype, alias

Creating new templates

To create a new documentation template, select **File | New | New Template** from the main menu of the Documentation Template Designer. The New Template dialog box displays the RWI meta-model for reports based on the project model and two kinds of templates:

- **Document Template**: for RTF, text, or non-frame HTML documentation
- **Frameset Template**: for HTML frame-based documentation

All of the topics discussed in this chapter and the next (Chapter 13, "Documentation Template Controls") are appropriate for document templates. Frameset templates for multi-frame HTML are covered in Chapter 14, "Creating Multi-Frame HTML Documentation."

The main menu of the Documentation Template Designer has four commands: File, Object, Stock, and Help.

File provides the usual file operations (creating new templates, opening or re-opening existing templates, saving changes, exiting). You should go here to access properties for template settings.

Object is identical to the right-click menu of the currently selected template "object" -- both display the available object operations. Template objects can be the bands that make up a template as well as the visible items in the bands. Different kinds of objects have different right-click menus, although all menus list **Properties** or **Area Properties**. The list of items on right-click menus for static sections, headers, and footers varies between the scope pane and the details pane. (There is no Object or right-click menu for headers and footers in the scope pane.)

Stock is for creating a new stock section or editing an existing one.

Setting template properties

You can access options for a template in the Documentation Template Designer by selecting **File | Properties** from the Designer's main menu. Figure 41 shows how the **Template Properties** dialog box displays the template settings on four tabbed pages: General, Page Settings, Formatting Styles, and Template Parameters.

- **General** properties are for defining the report title, picking the metatype of the root element, and toggling headers and footers. At the top of the page is a field for describing the template.

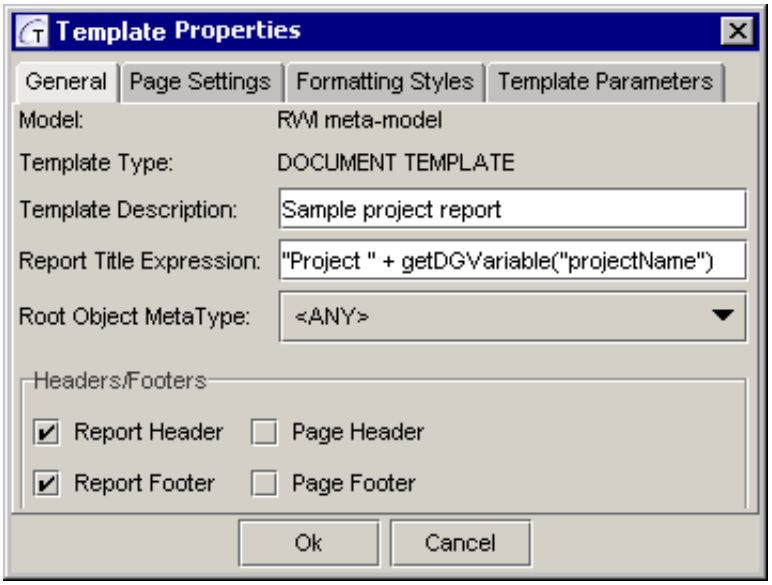
  The **Report Title Expression** is used for HTML documentation. DocGen evaluates this expression to put the title on the browser window.

  DocGen starts processing a template using a root element from the model of the type given by the **Root Object Metatype**. If this template is called from a different template, then the calling template supplies the actual root. Otherwise, the root element is the model, the current diagram, or the current package depending on how the documentation generation is started. Since the current element metatype must be consistent with the metatype of the current section, the Root Object Metatype limits where the template can be called.

Tip The default root metatype, `<ANY>`, is generic and consistent with all metatypes. You should use this default if you do not want to limit where the template can be used.

- **Page Settings** enables you to specify page size, margins, and landscape or portrait orientation. Measurements for the page size and margins are in inches.
- **Template Parameters** are formal parameters used for calling this template from another template. There are three fields for each formal parameter: the parameter's name, the description, and a default value (a string). The name is required. The description and default values are optional.

Figure 41 Template Properties dialog box



Designing the template body

The body of a newly created template consists of a generic element iterator and a static section nested within. It provides a minimal base for constructing the tree of sections. Every new section must be a sibling or a child of an existing section.

Creating, arranging, and resizing sections

To create a new section, right-click on an existing section and select **Insert Sibling Section** or **Insert Nested Section**, then select the type of new section from the resulting menu. If the existing section is not static, you can right click anywhere in the section. For static sections, you must right click on the portion of the section that lies in the scope pane to create a sibling section.

You cannot change the level of a section and you cannot move a section to a different parent. However, you can change the position of a section among its siblings. Right click the section and select **Move Up** or **Move Down** to reorder the section one position higher or lower.

To collapse or expand a section, click the "+" or "-" in the section's upper left corner of the scope pane.

To resize a section, position the cursor over its lower edge in the details pane. After the cursor becomes a double-ended arrow, click and drag the lower edge to its desired position.

Setting section properties

After creating a section, you can change its properties by selecting **Properties** (or **Area Properties**) from its right-click menu or from **Object** on the main menu. The resulting dialog box is organized into tabbed pages. The tabs vary according to the type of section. Element iterators, property iterators, and folder sections each have a tab for **Output Style**, and a tab for **Other** options. (**Settings** displays for a folder section)

Output Style

The Output Style tab is for specifying whether the documentation is to be in paragraph, text, or table format.

- **Paragraph Flow**: In this default format, documentation for each element constitutes a single paragraph.
- **Delimited Text Flow**: Delimiters separate the documentation for different elements. You must fill in the delimiter, and you can fill in an output style (a named style defined in the template properties) and font characteristics. There is a check box for printing headers and footers. There is another check box for suppressing formatting options.
- **Table**: Documentation for different elements is written to a table, with one row per element. Within each row, the different pieces of documentation are in different table cells. You can set border styles and cell paddings. There are two check boxes for RTF documentation: print a separate table header on each page and allowing breaking a table over successive pages.

Other options (Settings tab for folder sections)

The Other tab (Settings tab for folder sections) has options for setting the left indent and for specifying enabling conditions. Indents are relative to the indentation for the containing sections rather than the physical paper border.

An *enabling condition* can turn on or off different sections or controls within sections. Checking the **Disabled** box skips the entire section.

Additionally, the folder section offers a commentary field in its Settings tab where you can enter a descriptive string to identify the folder section in the template.

Enabling Conditions

Enabling conditions are boolean expressions for turning section processing on or off. If the condition is true, DocGen will process the section; if false, the DocGen will skip the section.

Enabling conditions typically have subexpressions that are properties of the metatype of a section or calls to special *DG functions.* (See Chapter 15, "Documentation Generator and Template Designer Reference" for a list of DG functions and variables.) The results may be joined together with logical operators (`==`, `!=`, `||`, `&&`) under the usual precedence rules. Here are three examples:

- `hasPropertyValue("$name", "problem_domain")`
- `getContainingNode() -> hasProperty("$interface")`
- `getParam("ShowImages") == "graphics" && getParam("Headers") != "show"`

Accessing model elements through iterator sections

Models are trees composed of elements such as diagrams, packages, classes, operations, and properties. When DocGen reaches an iterator section, it traverses the subtree of the model rooted at the current element. It applies the iterator to the children of the current element that match the iterator's metatype, recursing inside those children if instructed.

There are two kinds of iterators, element iterators and element property iterators. All iterators must have other sections nested within. Iterators may have optional headers and footers.

Element iterators

Element iterators provide a way of looping through elements of a model. Each element iterator has its own metatype, which must be consistent with the metatype of the iterator's parent's. For example, a section with activity diagram metatype may contain a swimlane iterator but it may not contain class or actor iterators.

When DocGen enters an element iteration section, it calculates a new current element according to the current element of the parent section and the metatype of the iterator. DocGen loops through an element iteration section using each possible new element as the current element for that iteration. The properties of an element iterator affect the way a new current element is calculated and how it changes during iterations.

If DocGen enters an iterator and finds no elements corresponding to the iterator's metatype, it will produce no documentation except for an optional header or footer.

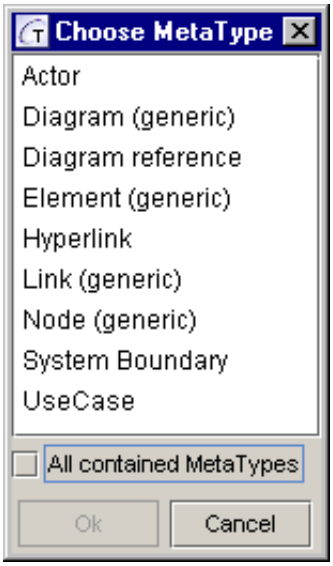
When you create a new element iterator, you must select its metatype from a list whose choices depend on the metatype of the parent section. If you check **All contained Metatypes**, the list expands to include the available metatypes of the items on the list as well. Figure 42 shows the available child metatypes for UseCase Diagram iterator.

Tip If you want an iterator to be able to access an entire model, choose *Package* or *Diagram (generic)* as the metatype.

After you create an element iterator section, you can change its properties: choose **Properties** from its right-click menu to bring up the Element Iterator dialog box.

Note You can change an iterator's metatype after you create it, but you should keep the new metatype consistent with its parent and any of its children.

Figure 42 Child metatypes for a Use Case diagram iterator



Element iterator scopes

Scope Options are for determining which elements of the model this iterator will document. Each iterator works over the subtree of the model that is rooted at the current element (the element that starts the iteration).

The top of the Scope Options tab has three buttons for limiting the part subtree that the iterator will apply to.

- **Default**: There are no special limitations. The elements considered depend solely on the element starting the iteration and the metatype of the iterator.
- **Customized**: The elements of the subtree under consideration are further limited by two expressions for DocGen to process. One expression describes which element to begin with; the other describes how to get the subsequent element from the current one. ("Formula and text controls" on page 209 has more information on constructing expressions.)
- **Programmed**: This limitation of the subtree under consideration works in conjunction with Together's open API. This is particularly useful for custom elements. When you choose Programmed, you must specify the name of the method that can access the elements and the class to which it belongs.

With the Default option, you can choose how DocGen should recurse the subtree rooted at the current element to search for elements. There are five different search strategies to find which elements of the subtree to document. You can reject (**no**), allow (**yes**), or partially restrict (**expr**) recursive and other kinds of searching. If you choose **expr**, you must enter an expression to define the restriction.

- **Recurse subpackages**: Search inside packages.
- **Recurse Subnodes**: Search inside elements.
- **Include Parent Element**: Visit the parent element (the current element, from which the iteration was initiated). Normally, an iterator goes through the contents of an element, not the element itself. This option is for documenting the containing element as well.
- **Visit Diagrams**: Search diagrams as well as packages.
- **Include Shortcuts**: Search shortcuts contained in elements.

The field at the bottom of the Scope Options page is for filter expressions. A filter expression restricts the search to elements that satisfy the filter properties. For example, `!hasPropertyValue("$name","<default>")` restricts the search to non-`<default>` diagrams or packages.

Note Filters and an enabling condition work at different levels of granularity. A filter works per iteration while an enabling condition works at a global level, turning on or off the entire iterator section.

Sorting

You can specify the order in which the elements are to be searched and thus documented on the Sorting tab of the Element Iterator dialog box. There are three options:

- **None**: the default choice.
- **Element Property**: sort the elements searched according to a property of the section's metatype. You must select the property from a list of properties.
- **Expression**: sort the elements searched according to a string-valued expression. You must enter an expression that DocGen can evaluate. (See "Formula and text controls" on page 209 for information on constructing expressions.)
- **Reverse scope order**: elements are always documented in ascending order. Check the **Reverse scope order** box to list elements in descending order.

Element property iterators

Element iterators traverse model elements. Element property iterators traverse element properties instead of elements. For instance, since an operation is a class property, you can use a property iterator to loop through all of the operations of a class.

A property iterator can reside inside an element iterator, folder, or property iterator. A property iterator must contain at least one static section, folder section, or call to a stock section or template. A property iterator may also contain an element iterator, or another property iterator.

You can select the properties of an element property iterator when you create it. The Element Properties dialog box has three tabs: Iteration Scope, Output Style, and Other. **Iteration Scope** lists the property to iterate over. There are three sets of properties choices:

- **Set of Properties**: These are properties that belong to the metatype of the parent element iterator. (See "Metamodel types" on page 190.)
- **All User-Defined Properties**: These are properties that are not described in the metamodel. The dialog box has two checkboxes:
  - **Exclude already iterated properties** omits properties that were already iterated for the current element.
  - **Iterate only unknown properties** includes only those properties that were not included in the metamodel.
- **Instances of a Single Property**: These are properties that can have multiple values, for example, @see or @author.
  - **Filter Expression**: Use this field to restrict the search elements that satisfy the filter properties.

A property iterator can iterate over multiple properties. Set the scope to **Set of Properties**. Use Ctrl+click to select multiple properties from the *Available properties* list, and then use the double-arrow button to move these properties to the *Selected properties* list. You can change the order in which the properties are documented by arranging the properties in the *Selected properties* list.

Folder sections

Folder sections group other sections together. A folder has at least one nested section, and it may have a header or footer. In that sense, folders are similar to element iterators, except that DocGen executes folders only once.
Folders inherit their metatypes from their parents. The sections nested within a folder must be consistent with its metatype. Folders provide a way to put section-level properties on their contents. This includes enabling conditions for toggling its processing on and off.

Both folders and iterators can have headers and footers. If the sections in a folder do not result in output, then the folder's headers and footers are ignored. Headers and footers of iterators are not ignored, even when no other output for the iterator is produced.

Reusing templates and stock sections

Static sections and iterators are critical for templates. Folders are conveniences for organizing template sections. Calls to stock sections and external templates are essential for reusing existing sections.

Calls to stock sections

Stock sections are reusable folders or iterators that reside in the template's collection of stock sections. They are not shared among different templates. When DocGen processes a call to a stock section, it is the same as if the called stock section were simply embedded at the position of the call.

Stock sections are especially convenient for frequently used constructs. You can insert a call to a stock section from any section whose metatype is consistent with the metatype of the stock section. Stock sections may contain calls to other stock sections as well recursive calls to themselves.

Creating calls to stock sections

To create a stock section, select **Stock | New** from the main menu of the template designer. The resulting dialog box displays a choice of appropriate sections. Once the stock section has been created, you can right-click the section to bring up its **Properties** options. The resulting dialog box has two tabbed pages: **Call to** and **Other**. **Call to** lists the possible stock sections and highlighting the name of the stock section that is actually called. **Other** has a field for the enabling condition, a field for the template parameter expression, and a field for setting the left indent. Check **Disabled** to skip the entire stock section.

Creating and editing stock sections

You can create a stock section from an existing folder or element iterator by right clicking on the section and selecting **Copy into Stock**. You can also create a new stock section without using an existing section by selecting **Stock | New | Folder Section** (for a folder root) or **Stock | New | Element Iterator** (for an iterator root) from the main menu of the Document Template Designer. In the resulting dialog box, you must enter the new stock section's name. If the root element for the stock section is an iterator, you must also select a metatype. You also have the option to designate the stock section to use an **Intrinsic Property Iterator**. If you check this flag, then this means that the only available iteration scopes for an element iterator are *customized* and *programmed*. For folder sections, this means that the *metatype* chooser tab is absent.

You can edit a stock section by selecting **Stock | Edit** from the main menu. The resulting dialog box displays the list of stock sections. Click to select, or Ctrl+click to select multiple sections. There are four options:

- **Edit** the selected stock sections. This command displays each stock section in its own tab of the Documentation Template Designer.

- **Delete** selected stock sections from the list of stock sections. (All calls to these stock sections are replaced by calls to "<none>.")
- **Copy** the stock sections into the clipboard.
- **Paste** the contents of the clipboard to create a new stock section. Paste adds copies to the list of stock sections, automatically generating their names (for example, Class, Class1, and so on).

To edit or display a single stock section from the template, select **Show Stock Section** from the right-click menu of a call to the stock section.

When you add a new stock section or edit an existing one, the Document Template Designer window displays a new tab for the stock section. Right-clicking the tab enables you to close or rename the stock section.

Calls to template sections

With a call to a template, DocGen can produce documentation using a different template without terminating the current one.

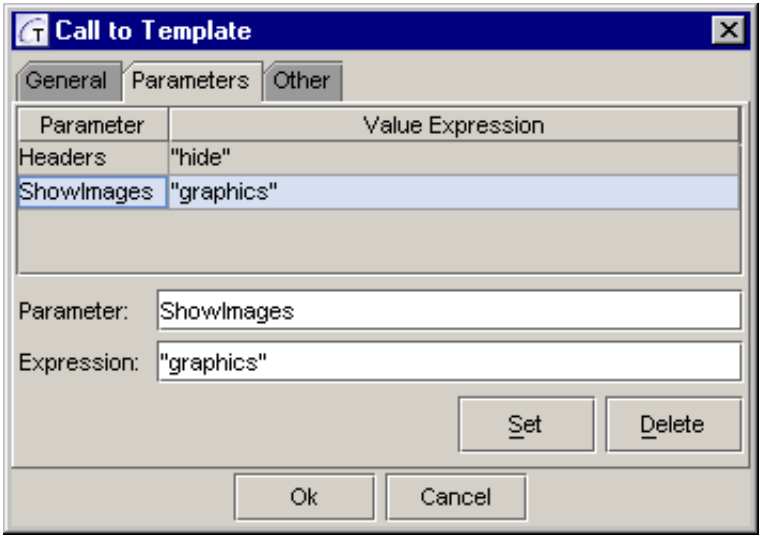
After creating a call to template section, right-click the new section and select **Properties**. This displays the Call to Template dialog box, where you specify the name of the called template. Clicking on the button to the right of the file name text field opens a dialog box for selecting the actual called template.

The Call to Template dialog box gives a choice of where the output for the called template goes:

- **Separate file**. This is important for generating multi-frame HTML documentation consisting of separate HTML documents that are extensively linked together.
- **Common stream**. The called template behaves like a stock section.

When a template is called, the current element of the calling template becomes the root element of the called template. A calling template can pass additional information to the called template through template parameters. Figure 43 shows the tab on the Call to Template dialog box for specifying the parameters and assigning actual values to them. The formal parameters are options of the called template. (See "Setting template properties.")

Figure 43 Actual parameters for a Call to Template Section



A called template can access actual parameter values through the function `getParam`, which has the following signature:

```
String getParam(String parameterName)
```

Parameters are especially useful for enabling conditions. Here is an example of an enabling condition based on Figure 43:

```
getParam("ShowImages") == "graphics"
```

For more information, see "Enabling Conditions" on page 195 and "Formula and text controls" on page 209.

Calls to templates make it possible to construct a library for generating documentation for particular model elements (class, actor, use case, and so on). By calling pre-designed library templates, you can quickly construct templates for more general reports (such as the built-in template `ClassReport.tpl`) intended for both multi-frame HTML output and RTF printable documentation.

Chapter

13

Documentation Template Controls

The documentation generator uses Together projects and templates to produce project reports. This chapter discusses *controls*, which are the items in documentation templates that determine the contents of reports. Controls are the commands to the DocGen engine for placing data into reports.

This chapter assumes that you understand the concepts discussed in the previous chapter, "Designing Custom Documentation Templates." The contents of this chapter include:

- "Using static sections, headers, and footers" on page 203.
- "Creating controls and setting control properties" on page 205.
- "Managing the display of output" on page 210.
- "Hyperlinking controls to element documentation" on page 212.

Using static sections, headers, and footers

Documentation templates consist of headers, footers, and body sections. The body of a documentation template is organized as a tree of sections. Static sections, calls to stock sections, and calls to templates are the leaves on the tree -- they can have siblings but they cannot have other sections nested within. (See "Body sections and the DocGen engine" on page 188.)

Of the six kinds of body sections, only static sections contain controls for producing actual output. Headers and footers can also contain controls. Folders and iterators, which cannot directly contain controls, must have at least one static section nested somewhere within.

Creating and deleting static sections, headers, and footers

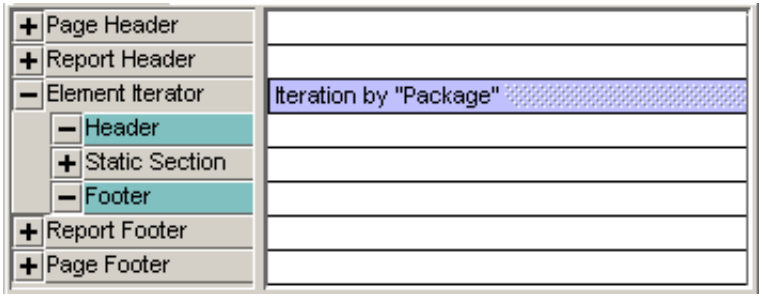
To create a static section, right click an existing section and choose **Insert Nested Section | Static Section** if the existing section is a folder or iterator or for any existing section, choose **Insert Sibling Section | Static Section**. Alternatively, select the existing section in the details pane and choose **Object** from the main menu.

The metatype for a static section is the metatype of its parent. ("Body sections and the DocGen engine" on page 188 has an introduction to metatypes.)

Report and page headers and footers lie outside the body of the template. You can turn report and page headers and footers on or off in the template options (use **File | Properties** from the Documentation Template Designer main menu).

Iterators and folders can also contain headers and footers. To add a header or footer to a folder or iterator, right click the section, and choose **Add Header** or **Add Footer**. Figure 44 shows the aqua coloring of body section headers and footers in the scope pane.

Figure 44 Section headers and footer coloring



Each header, footer, and static section has two different right-click menus, one for the scope pane (on the left) and the other for the details pane (on the right). The right-click menus on the details pane are for setting area style and format characteristics and for inserting controls.

The right-click menus on the scope pane vary among headers, footers, and static sections. Items on those menus include the following:

- **Delete**: for headers, footers, and static sections with siblings only. Delete for static sections without siblings is disabled.
- **Insert Sibling Section**: for headers and static sections only. The new section goes immediately below this header or section.
- **Move Up**, **Move Down**, **Copy**: for static sections only. **Copy** creates a copy of the section in the clipboard. You can paste from the clipboard when you insert a new section.
- **Properties**: for static sections only. Static sections have only one property, which is an enabling condition. (See "Enabling Conditions" on page 195.)

Setting area properties

To set area properties for headers, footers, and static sections, choose **Area Properties** from the right-click menu of the details pane. Selecting Area Properties displays a dialog box with three tabbed pages:

- **Settings**: contains check boxes for page settings (such as "start a new page with this section") and for suppressing formatting.
- **Hypertext Target**: for inserting anchors and bookmarks. (See "Hyperlinking controls to element documentation" for more details.)
- **Other**: for associating formatting styles with an area, setting style name expressions, and using a control delimiter.

Creating controls and setting control properties

There are six different kinds of controls. To create a new control in a static section, header, or footer, choose **Insert Control** from its right-click menu in the details pane followed by:

- **Label**: for static text
- **Image**: for graphics (project diagrams or other images)
- **Panel**: for a container for other commands
- **Formula**: for project-specific data (necessary when the data cannot be accessed with simple data controls)
- **Data Control**: for project-specific data
- **Include Text**: for including text from external files

When you insert a new control, the Documentation Template Designer displays a dialog box for setting the control's properties. The template shows each control as a shaded rectangle in the details pane.

You can change the properties of a control after it is created by right-clicking the control and choosing **Properties**, or by selecting the control and choosing **Object | Properties** from the main menu of the Documentation Template Designer. The result is a dialog box organized into tabbed pages for the properties settings.

Labels, images, and panels

The simplest kinds of controls are labels, panels, and images.

Labels

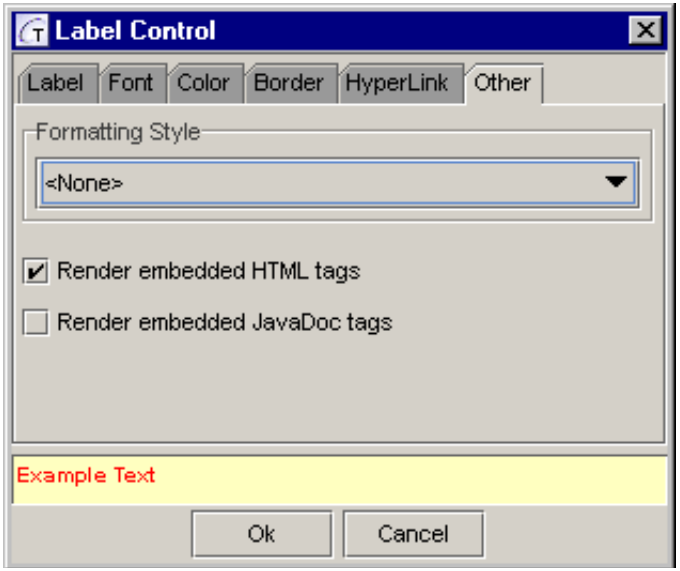
A label control generates static text that is independent of its containing section. The text does not depend on the metatype of the section or where the section

belongs in the template. Placing identical labels in a header and a static section results in the same output as long as the header and static section are not skipped. Label properties include the label's text, style (font, color, and border), and if and how to hyperlink the output. The section, "Text format settings" on page 211 has more details on styles.

Figure 45 shows the **Other** tab of the Label Control dialog box. There is a menu for choosing a named style from the template options. The checkbox for *Render embedded HTML tags* determines whether to interpret any HTML tags in the label as HTML tags or simply show them as text.

On the **Other** tab, you also have the option to *Render embedded Javadoc tags*. For example, enter the following text, `@link Class1`, on the **Label** tab, and check *Render embedded Javadoc tags* on the **Other** tab of the label control. You also need to mark the area that you wish to reference by the hyperlink as a location of the element's main documentation. To do this, you open the **Area Properties** for the static section under *Iteration by Class*, select the Hypertext Target tab, and check *Start of the current element's specific documentation*. Whenever you generate documentation, the label displays as a hyperlink to the documentation for `Class1`.

Figure 45 Label properties



Images

An image can be external to the project or it can be a project diagram. The Image tabbed page on the Image Control dialog box has a **Type** menu for selecting which. There are two choices:

- **Static (URL)**: Used for a file (gif, jpeg, or suitable image file type). With this choice, you should specify the name of the file in the *File* text field at the top of the page. You can fill the name in directly or click the command button to the right of the text field to open a file selection dialog box.
- **Static (Resource):** Use the *Image Resource Expression* field and enter a path to the image file. For example, "`..\modules\com\togethersoft\modules\robustness\images\tv-boundary.gif`"
- **Diagram**: The image is a diagram from the project. If the containing section metatype is a diagram, DocGen will produce an image of the diagram of the current element when it processes the section. If the containing section metatype is not a diagram, choosing Diagram results in no output.
- **Element's Small / Large Icon:** Inserts a small or large icon representing the iterator's metatype.

Image controls can have borders. Use the **Border** tab of the image control to define borders around an image. Image controls can also be hyperlinked in the same way as label controls.

Panels

A panel is simply a container for other controls. Panels are convenient for grouping controls together to provide a uniform style and precise alignment. You can set the panel's background color, border, and style.

To insert a new control into a panel, use the panel's right-click menu and choose **Insert Control | <control type>**. Alternatively, click the panel, and choose **Object | Insert Control | <control type>** from the main menu. If a copy of a control is in the clipboard, you can choose **Paste** from the **Object** main menu or the panel's right-click menu to insert it into the panel. You cannot drag a control into a panel from outside of the panel.

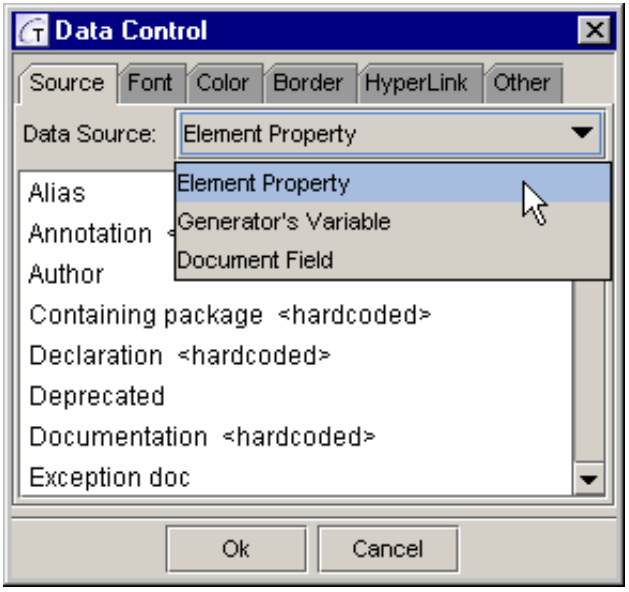
Data controls

Data controls provide the major mechanism of placing data from a project into a report. When DocGen processes a data control, it obtains the actual data from the current model element. (See "Body sections and the DocGen engine" on page 188 for a review of current model element.)

Data Control Sources

The source of data for a data control can be DocGen variables, the generated report, or (most frequently) the project. When you create a data control, you must specify its data source. Figure 46 shows picking the data source from the drop-down menu in the Data Control dialog box.

Figure 46 Data Control dialog box



The menu includes three choices:

- **Element Property**: A property of the current element. The Data Control dialog box displays a list of every property belonging to the metatype of the current model element. Metatype properties are defined in the metamodel files. (See "Metamodel types" on page 190.)

  If the root object metatype (for the entire template) is `MODEL` or `<ANY>`, the list of Element Properties for report and page headers and footers is empty.

- **Generator's Variable**: A variable used by DocGen. You can use this in report headers or footers to insert the project name or the date and time the report is created.

  If the data control is inside a property iterator, the list of Generator's Variables includes the property's characteristics (value, name, and so on).

- **Document Field**: A field of the report such as page number or bookmark. You can select Document Field to insert page numbers and number of pages into page headers or footers. The Document Field list is empty for report headers and footers.

The Data Control dialog box has tabbed pages for formatting, hyperlinking, and substituting default text for missing data.

Displaying custom properties

Data controls can generate documentation for custom properties if you make the appropriate modifications.

To display custom properties:

1. Add the property.
2. Edit the file, `MetaModel.mm`, by putting the property in two places:
   - In the list of properties at the beginning of the file.
   - In the set of properties belonging the appropriate metatype for that property.

When you create a data control in a section whose metatype is that of the new property, the Data Control dialog box lists the new property among the Element Property sources.

The next section on Formula Controls discusses an alternative way of displaying custom properties.

Formula and text controls

Formulas provide a way to place data into a report that DocGen calculates when it processes the control. You must enter the formula that DocGen evaluates to calculate that output. Both formula controls and text controls rely on such formulas.

Formula controls

A *formula* is an expression that DocGen can evaluate to a string. The expression can be a combination of string literals, *DG variables*, and *RWI functions* from the read-write interface of Together's Open API.

DG variables are special variables that are available to DocGen at runtime when it is producing a report. DG variables include items such as current element, the date and time, and template parameters. A complete list of DG variables and RWI functions is in Chapter 15, "Documentation Generator and Template Designer Reference."

When you insert a formula control, you must enter the formula in the Formula Control dialog box. Formulas use C++ style syntax, with double quotes for string literals, `+` for string concatenation, and `->` for calls to functions via pointers. The following example from a section with class metatype puts *Package* followed by the name of the containing package into the report.

```
"Package " + getContainingPackage() -> getProperty("$name")
```

The next example is a formula control inside of a section with a generic class metatype. It puts *Interface* in the report if the current element is an interface and *Class* if it is not.

```
if (hasProperty("$interface"), "Interface", "Class")
```

The remaining properties for a formula control are similar to those for a label control.

To display custom properties in the generated report, you can follow the instructions outlined in the previous section on data controls. Alternatively, you can use a formula control to specify the property. This does not require editing the metamodel file. The function call `getProperty(property_name)` gives access to any property, whether it is included in the metamodel or not.

Include text controls

Include text controls are used for copying text from other files into a template. When you insert an include text control, you must enter an expression for the location of the text file. The expression can be hard coded as a string literal, or it can use a formula as described above.

Include text controls have formatting properties identical to those for formula and label controls.

Managing the display of output

The positioning and properties of controls effect their associated display in the generated reports.

Moving and resizing controls

The Documentation Template Designer displays a newly created control as a rectangle positioned at the insertion point. You can move the control to change where its output appears in the generated documentation. You can also modify the size of the rectangle to determine the approximate size of the region for the output. (Precise positioning and sizing is not possible.) Increasing the rectangle size is especially important for a label for which the default size is not large enough to allow its entire text to be displayed in the report.

To move a control within a section:

1. Select the control. Notice that the mouse pointer changes to a cross with double-ended arrows.
2. With the control selected, drag the control to the desired position within the section.

To resize a control:

1. Place the mouse pointer on either the right or left edge of the rectangle. Notice that the mouse pointer changes to a double-ended arrow.
2. Drag the edge of the rectangle to a desired size.

Tip You can move multiple controls within a section at the same time. To select multiple controls, use Ctrl+click.

You cannot move a control from one section to another by dragging it. Instead, you must **Cut** using the control's right-click menu and **Paste** using the target section's right-click menu.

Text format settings

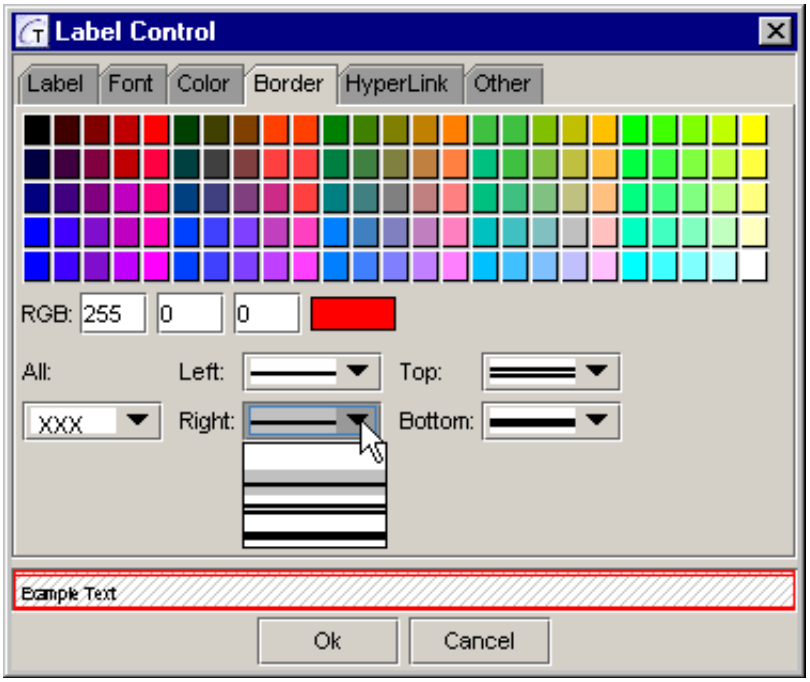
Properties for controls include tabs for formatting the presentation of text: **Font**, **Color**, and **Border**. Each page has a preview box at the bottom for evaluating the format settings.

On the **Font** tab, you can select the font family, size, style, and alignment within the rectangular area of the generated output.

On the **Color** tab, you can select the text and background colors by clicking the color chips or filling in the RGB values. Check the transparent checkbox to the right of the Fill color to make the background transparent. A hatched background in the preview indicates a transparent background.

On the **Border** tab, you can specify whether you want a border, and also determine its style, and color. The border styles are on drop-down lists, as shown in Figure 47.

Figure 47 Setting border properties



Aligning controls

You can move a single control within a section to determine the positioning of the associated output and its ordering within the output for the entire section. You can also move multiple controls by first selecting them with Ctrl+click.

If you select two or more controls in a section, you can use the control's right-click menu to uniformly align them within the section. The alignment options are *Left Side, Right Side, Top Side,* and *Bottom Side*. You should take care when applying multiple alignments since it is possible for controls to overlap.

You can also use the right-click menu to apply uniform sizes to the controls' enclosing rectangles: *Make same width, Make same height,* and *Make same size*.

Note There is no simple undo for changes in alignment or size. When you change alignment or resizing, you must manually readjust the controls to return them to the former status.

Hyperlinking controls to element documentation

A *hypertext link* connects a *link reference* (starting point or source) to a *link destination* (*target*). The link reference is a text or image in the HTML document. The link destination is a file (usually an HTML document or an anchor in an HTML document).

Document templates support both references and targets. Link references are properties of controls. Link targets are properties of static sections, headers, and footers.

Making a target from a static section, footer, or header

Any generated output that contains an anchor or bookmark can be a link target. Documentation templates have facilities for inserting anchors at the "main documentation" of model elements.

You can insert anchors for static sections, headers, and footers. Start with **Area Properties** on the section's right-click menu and go to the **Hyperlink** tabbed page. The page displays three items:

- **Expression for Target Bookmark Selector**: This is a text field.
- **Start of the current element's specific documentation**: This is a checkbox to identify the output of this section as the "main documentation" for the current element. When DocGen processes the section, it inserts a hypertext anchor or bookmark into the output, automatically generating its name. DocGen recognizes this section as the *element's main documentation.*
- **Expression for Documentation Subject Selector**: This is a text field also used for documentation for the current element.

Linking a control to a target

When you define the location of a model element's main documentation, you can specify hyperlink references to it for any control that is not a panel. The control can be in the same template as the main documentation or it can be in a different template.

To hyperlink a control to a target, select **Properties** on the control's right-click menu. Then go to the **Hyperlink** tab. Figure 48 shows the Hyperlink tab.

There are three choices of hyperlink target types:

- **File Link:** You must fill in the path to the file. If the hyperlink target is a bookmark in the file, you must supply that as well.
- **URL Link:** You must supply the URL.
- **Link to Element's specific Doc:** You must identify the element whose documentation is to be the target in the text field for *Expression for RWI-Element*.

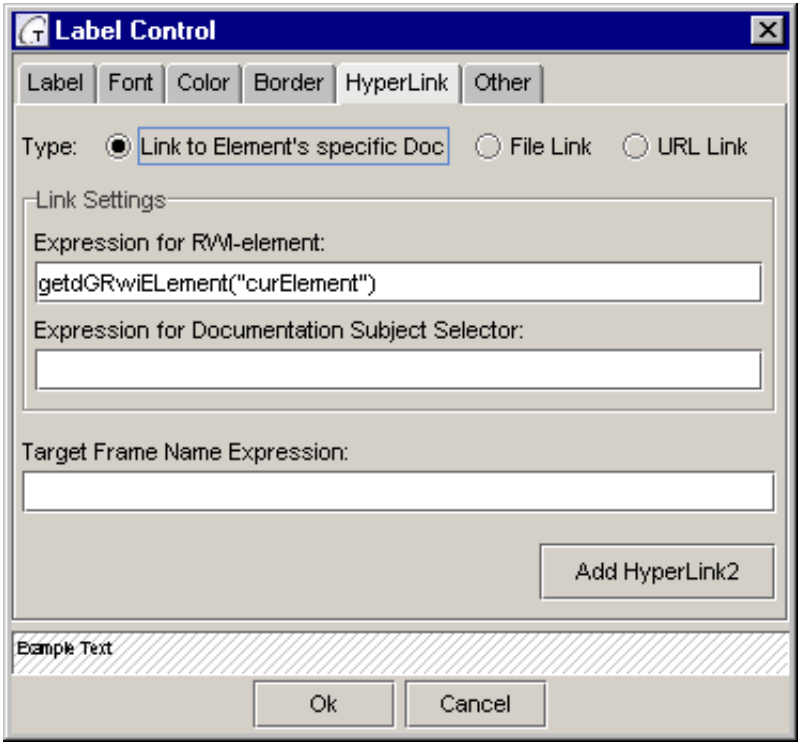
DocGen evaluates the *Expression for RWI-Element* to determine which element's documentation is the hyperlink target. For hyperlinking to the current model element, you can use the following expression:

```
getDGRwiElement("curElement")
```

This is the same expression as in Figure 48. Expressions can be more complicated, such as:

```
findElement(getDGRwiProperty("curPropertyInstance")->
            getSubproperty("$referencedElement"))
```

Figure 48 Hyperlinking a label to an element's main documentation



"Creating hypertext links (advanced)" on page 221 has additional information on documentation hyperlinks.

Chapter

14

Creating Multi-Frame HTML Documentation

Together can generate simple HTML, RTF, text, PDF, and multi-frame HTML documentation. This chapter discusses how to use the Documentation Template Designer to create the templates for multi-frame HTML documentation. It assumes that you understand the material covered in the two previous chapters, Chapter 12, "Designing Custom Documentation Templates" and Chapter 13, "Documentation Template Controls."

The contents of this chapter are:

- "Multi-frame HTML document template basics" on page 215.
- "Defining the frameset structure" on page 216.
- "Designing the frameset template body" on page 218.
- "Creating hypertext links (advanced)" on page 221.

Multi-frame HTML document template basics

Multi-frame HTML documentation divides project reports into frames to give multiple views within the same browser window. Multi-frame HTML documentation consists of two kinds of HTML files:

- A collection of HTML files to define the content for each frame
- A *frameset* file to specify the layout of frames

The frameset file is the starting point for viewing the documentation. It has hyperlinks to the HTML files for its frames.

Frameset document template organization

A frameset template consists of two major parts. One part describes the frameset file. The other part, which is the body of the frameset template, contains calls to the templates that provide the contents of the frames.

Every frameset template is designed for a particular type of model element (a metatype) for which it can produce specific documentation. For example, you can design a frameset template for a class, an actor, or for an entire model. This is the same as for ordinary document templates. (See "Metamodel types" on page 190.)

Creating frameset templates

To create a new frameset documentation template, choose **File | New | New Template** from the main menu of the Documentation Template Designer. In the resulting dialog box, select the model type (**UI-Builder Model** or **RWI Metamodel**) and select **Frameset Template** for the **Template Type**.

Once you create a frameset template, you cannot convert it into a document template or vice versa.

Note By convention, the names of frameset templates end in "Frames." For example, "ProjectReportFrames.tpl" and "ClassReportFrames.tpl," which are part of the Together installation.

Defining the frameset structure

The *frameset structure* of a multi-frame HTML document describes how the frames are organized within the browser window. You can define the structure in a frameset template through the template properties. Select **File | Properties** from the main menu of the Documentation Template Designer. In the resulting **Template Properties** dialog box, go to the **FrameSet Structure** tabbed page as shown in Figure 49.

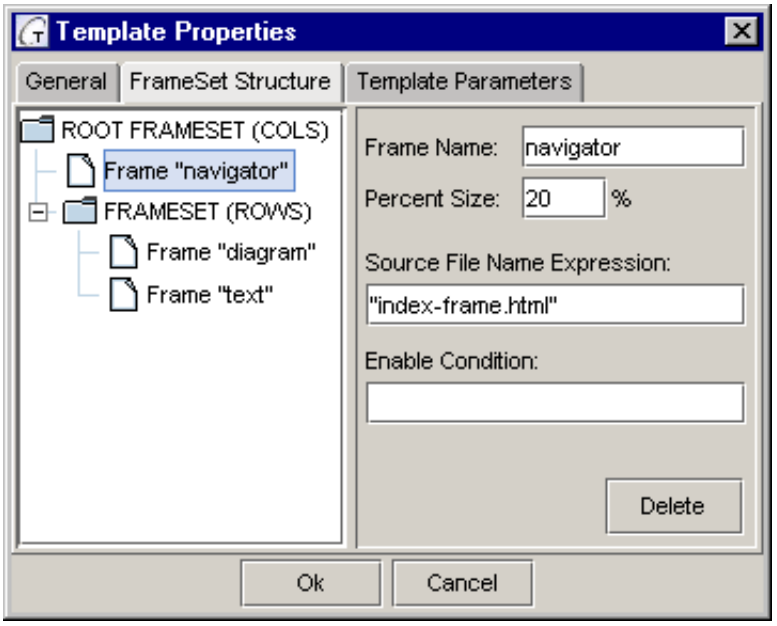
The FrameSet Structure tab shows the frameset as a tree on its left panel. The tree has two kinds of nodes:

- *Frame:* a frame to display a single HTML document. This corresponds to the <frame> tag in HTML. Frame nodes are leaves in the structure tree. Figure 49 shows three Frames: `navigator`, `diagram`, and `text`.
- *Frameset:* a container for additional frames or framesets. This corresponds to the <frameset> tag in HTML. In the tree, a frameset is the parent to the child frames and framesets that it contains. The tree in Figure 49 has two framesets and three frames.

In the tree, nodes with children have folder icons. Nodes without children have page icons. The root-level node is always a frameset.

The right pane of the FrameSet Structure tab shows the properties of the currently selected item in the tree.

Figure 49 Frameset structure



Specifying frameset properties

The topmost property of the frameset is its **Layout**, with radio buttons for selecting **Columns** or **Rows**. A frameset with a row layout divides its window (HTML frame) into rows, with one frame per row for each of its children. A frameset with a column layout divides its window into columns, with one frame per column for each child.

The frameset structure of Figure 49 has a root frameset organized into columns. The `navigator` frame is the left column. The right column is an additional frameset, which is divided into two rows, one for `diagram` and one for `text`.

You can assign a **Percent Size** to each frameset child to determine the percentage of the frameset's total space to be allocated to the child. For example, the width of the `navigator` frame of Figure 49 is 20% of the root window. The total of the sizes of a frameset's children should be 100%. Otherwise, the browser will decide the sizes for the children when it displays the documentation.

A frameset has an optional enable condition, which determines if the frameset is to be skipped or included when DocGen generates the frameset file. This condition is identical to that for body sections. (See "Enabling Conditions" on page 195 for more details.)

Specifying frame properties

The DocGen engine translates the **Frame Name** into the name parameter (the name of the frame as a target) of the corresponding <frame> tag. You can use that name in a hyperlink to load the referenced document into the frame window. The tree in the left pane of the Frameset Structure tab shows the Frame names.

The DocGen engine evaluates the **Source File Name Expression** expression to determine the name of the HTML file that will be initially loaded into the frame. The section "Assigning a target frame to a link reference" on page 221 has additional details of this feature.

Enable Condition and Percent Size for a frame are identical to those for a frameset.

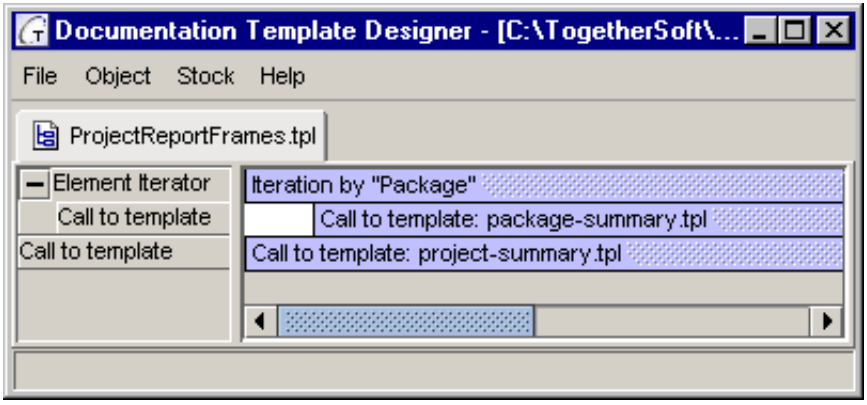
Designing the frameset template body

The body of a frameset template determines the documents that DocGen will create (or use) to load into the frames of the resulting project report.

Frameset template body organization

The body of a frameset template is similar to the body of an ordinary document template. Figure 50 shows the "ProjectReportFrames" template, included with your Together installation.

Figure 50 Frameset documentation template



A frameset template body can contain any number of iteration sections (element iterators and property iterators), folder sections, and stock section calls. However, static sections and headers and footers for folder sections and iterators are prohibited. Calls to template sections replace static sections to produce the actual output.

Note The body of a frameset template produces no separate output file.

Frameset template processing

When DocGen processes a frameset template, it produces the frameset HTML file as well as the separate HTML files for the frame content.

DocGen begins processing a frameset template at its body. When it encounters a call to a template section, the DocGen engine suspends the current template execution, loads the called template, and processes it to produce a separate HTML document. The root element for the called template is the current model element of the calling template. After it completes processing the called template, DocGen resumes executing the calling template.

After DocGen completes processing the body of the frameset template, it produces the special HTML frameset file. This file corresponds to the frameset structure specified in the template properties. The name of the frameset file matches the name of the frameset template. It is the starting point of the generated documentation.

Call to template section properties

The section properties of a call to template determine how the output for a template call can be used. With multi-frame HTML documentation, call to template sections typically generate separate files that can be loaded into a frame of the resulting HTML project documentation.

Note You can set the name of the file to be loaded into a frame through the Frameset Structure tab of the template properties. You can set the name of the output file generated from a call to template section through the section properties.

To access the properties of a call to template section, select **Properties** from the section's right-click menu.

The General tab of the Call to Template properties dialog box has the output settings. If the output generated from the template is to be loaded into a frame, you should select **Separate File** from the radio buttons at the top of the page.

Figure 51 shows the properties of the second call to template section from the "ProjectReportFrames" template of Figure 50.

Naming the generated document

Output File Name Expression is the name of the document. This expression is for the document name only; it should not include the file path.

If a particular call of a template is to be iterated many times to produce multiple documents, you should derive the output document name from the properties of the current model element. You can use the expression `getProperty("$name")` to get the name of the current model element.

If the Output File Name Expression is blank, DocGen names the generated document according to the name of the called template.

Naming the output directory

Output Directory Expression is the path to the destination directory of the generated document. This path is always relative. You should define it according to the following conventions:

1. If the calling template is a frameset template, the path is relative to the destination directory for the entire documentation.
2. If the calling template is a document template, then the path is relative to the location of the document that is generated by the calling template.
3. The right slash character "/" is the name-separator for the path.

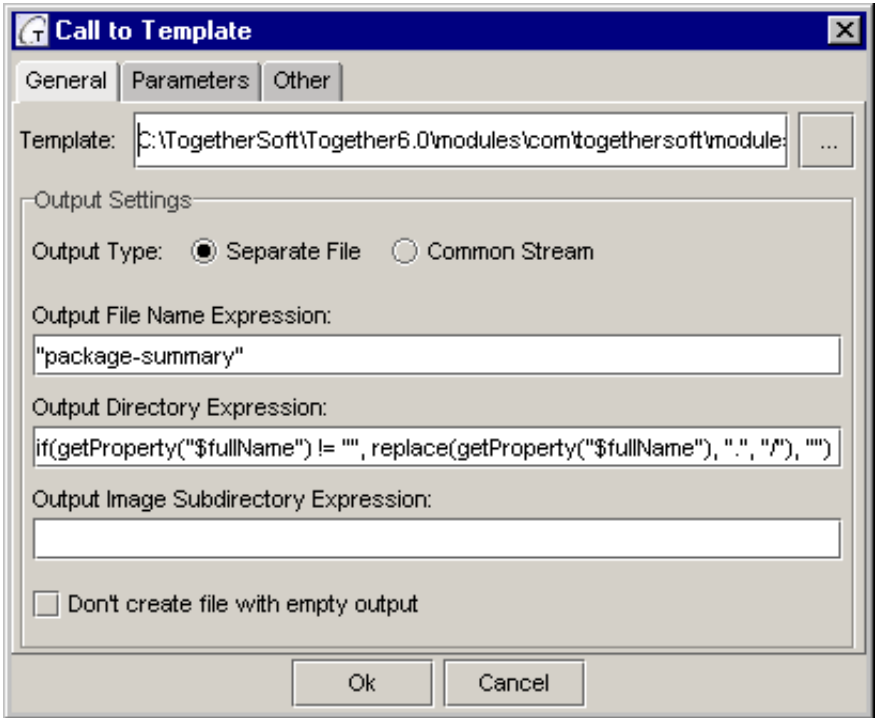
If the expression contains directories that do not yet exist, DocGen will create them when it processes the template.

The template in Figure 51 sets up a directory structure for the documentation to mirror the package structure of the model by having this expression as part of the path.

```
replace(getProperty("$fullName"), ".", "/")
```

The **Output Image Subdirectory Expression** is similar to the Output Directory Expression, except that it is meant for image files rather than HTML files.

Figure 51 Hyperlink for a call to template section



Creating hypertext links (advanced)

Multi-frame HTML documentation requires hypertext links. A hypertext link connects a link reference (source) and a link destination (target). The link reference is a piece of text or an image. The link destination is a file or an anchor in a file.

"Hyperlinking controls to element documentation" on page 212 discusses hypertext links in the context of ordinary document templates. This section expands that discussion.

Recall how the Documentation Template Designer supports references and targets:

- Link references are properties of controls. To access those properties, select **Properties** from the control's right-click menu. Use the HyperLink tabbed page of the resulting dialog box.
- Link targets are files, URLs, or properties of static sections, headers, and footers. To access those properties, select **Area Properties** from the section's right-click menu. Use the Hypertext Target tabbed page of the resulting dialog box.

Assigning a target frame to a link reference

By default, browsers load a link target in the same frame as the page containing the link reference. A `target` parameter in the HTML hyperlink tag changes that default behavior to load the target file into a named frame.

The Documentation Template Designer allows you to set the target frame to a named frame. To change that target for a control that is a link reference, go to the HyperLink page of the control's properties. Use the text field of the **Target Frame Name Expression** to enter an expression for the name of a frame window defined in the frameset structure.

Targeting an element's "specific" documentation

"Hyperlinking controls to element documentation" on page 212 explains how to simplify creating hypertext links using the notion of an element's "main" documentation. The advantage of this is that DocGen makes all necessary calculations and markups for the hyper-references.

It is occasionally necessary to provide link references to several different documents (or locations in HTML files) created with the same model element. For example, along with the main documentation file created for a package, there could be a different HTML document that simply lists all classes in the package. If this listing document were in a separate "navigation" pane, it would serve as an index for the package. Clicking the package on a diagram (or in some more general text) could load that listing document in the navigation frame.

The Documentation Template Designer enables you to target different documentation locations generated by the same model element.

Follow the same steps as for targeting the location of an element's main documentation:

1. Invoke the **Area Properties** for the template area where the location starts.
2. Choose **Hypertext Target** tab of the Area Properties.
3. Check **Start of the current element's specific documentation**.
4. Fill in an expression for **Expression for Documentation Subject Selector**.

After identifying specific documentation in this manner, you can define link references to that documentation in the same way as for an element's main documentation.

To create such a link reference for a control:

1. Invoke the **Properties** for the control.
2. Choose **HyperLink** tab of the Properties.
3. Select **Link to Element's specific Doc** from the radio buttons at the top.
4. Fill in the **Expression for RWI-Element** text field to determine which element's documentation is the link target. (See "Formula controls" on page 209 for a discussion of RWI elements.)
5. Fill **Expression for Documentation Subject Selector** to match the expression for template area described above.

Let us resume our previous example of creating an index for a navigation page. To make link references to the package's index document, we could mark the first area of the template for this document as the Start of the current element's specific documentation, using `"packageIndex"` as the Subject Selector. We would then enter `"packageIndex"` as the Subject Selector for any link references to this document.

Note The main documentation of an element is not merely an element's specific documentation with empty subject selector. Only the element's main documentation can be referenced from JavaDoc link references (those with the `@link` JavaDoc tag) embedded in the text returned by some RWI-properties. You cannot specify subject selector for these references.

Image mapping diagram elements

You can put an image control in a static section to include an image of a diagram in the generated document. DocGen will create an image only if the current model element represents a model diagram when it processes the section.

Image controls are similar to other controls in that their properties include a Hyperlink tab. If the image is not a diagram, the Hyperlink tab enables you to create an ordinary link reference. However, when the image control is for a diagram in the model, the same definition in the HyperLink tab creates link references for all model elements depicted on the diagram -- the resulting image

will be “image mapped.” To create the image map, you must enter all expressions in the link reference definition relative to the RWI-element returned by the call `getDGRwiElement("diagramMapElement")`.

DocGen generates the image map by iterating through elements of the diagram, substituting the variable `diagramMapElement` with every diagram element, calculating a link reference for it, and inserting the link reference into the image map.

Creating compound link references

Link references in multi-frame documentation may have multiple targets. Clicking on such a reference could simultaneously load two different documents in two different frames. For example, suppose a diagram element represents a package. Clicking on this element could load the image of the package diagram in one frame and the main (textual) documentation for the package in another.

To program such dual targets, you can define two different link references for a single control. Go to the HyperLink tab of control’s properties dialog box click the **Add HyperLink2** button at the bottom. Then you can specify the second link reference definition.

Javadoc link references

Javadoc References (or *JDRefs*) are the expressions associated with Javadoc tags such as `{@link}` and `@see`. You can use them to create link references inside documentation text (`{@link}`) as well as with some other documenting tags. DocGen can convert JDRefs into real hypertext links.

Each JDRef should conform to the rules described in the standard Javadoc documentation. There are three types of Javadoc references.

1. *element reference* refers to an element of the model (method, class, package, etc.). The general form of an element reference is: `package.class#member label`, where `package.class#member` is the referenced model element and `label` is optional text to be displayed with the link. (If `label` is omitted, the name of the referenced element is displayed.) DocGen can convert each element reference into a hyperlink to the main documentation of the element.
2. *url reference* represents a link to a relative or absolute URL. The general form of a url reference is: `<a href="URL#value">label</a>`
3. *text reference* has the form `"string"` (a text string in double-quotes). A text reference is simply information that does not represent a hyperlink.

Converting JDRefs into hyperlinks

A JDRef appears in one of two forms:

- inside `{@link}` tags embedded in documentation text. The JDRef is the value of the `$doc` property and other Javadoc element’s properties.

- as the value of some Javadoc element's properties such as `see`.

The Documentation Template Designer provides conversions for both cases. You need to specify the conversion in the properties of the control.

Converting {@link} tags

Only a text control (label control, data control, or formula control) can generate documentation text. To convert `{@link}` tags to hyperlinks, access the control's properties from its right-click menu. Then go to the **Other** tab and check **Render embedded JavaDoc tags**.

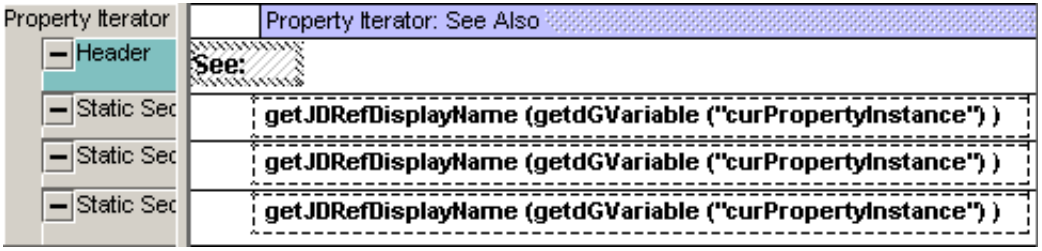
Converting the value of an element's property

Converting a value of an element's property to a hyperlink is more complicated than converting an `{@link}` tag. The conversions require using one of these documentation generation functions:

- `getJDRefType()`
- `getJDRefDisplayName()`
- `getJDRefElement()`
- `getJDRefURL()`

The remainder of this section is an example to illustrate how to display the values of the `see` properties with its associated hyperlinks. The template body for accessing those values is shown in Figure 52.

Figure 52 Iterator to display the see property values



The example uses a property iterator to go through the instances of `see` since `see` can potentially have several values. The property iterator contains three static sections that correspond to the three types of Javadoc references. Each static section has its own enable condition, which activates it for the appropriate JDRef.

The following enable condition activates the first static section for element type JDRefs only:

```
getJDRefType(getDGVariable("curPropertyInstance")) == "element"
```

Each static section area contains a formula control and nothing else. All formula controls use the same expression:

```
getJDRefDisplayName(getDGVariable("curPropertyInstance"))
```

The value of the expression is the text that will be displayed in the documentation.

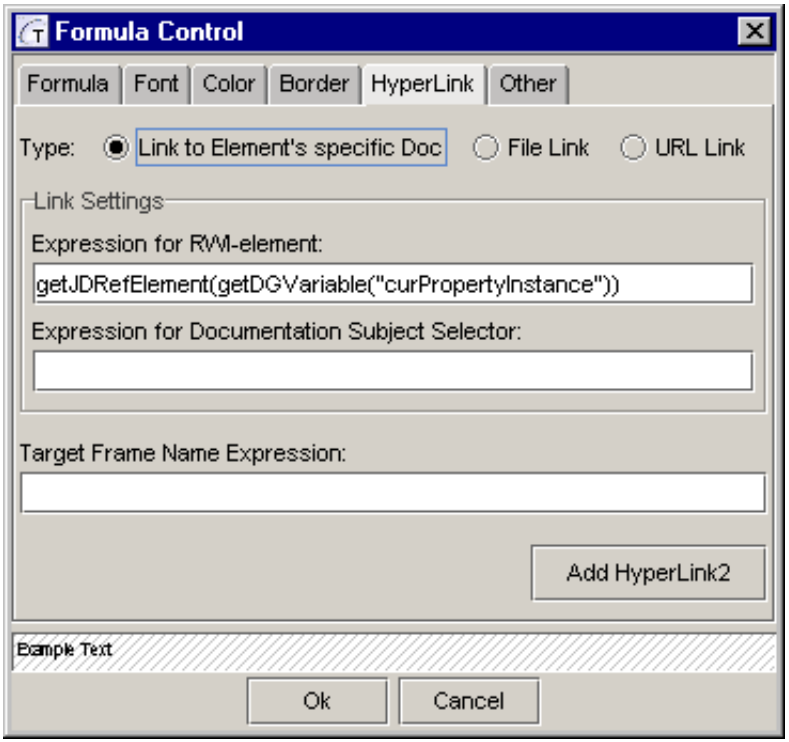
Figure 53 shows the HyperLink tab in the properties dialog of the formula control for the first static section. Note the value for Expression for RWI-element.

The function `getJDRefElement()` returns an RWI-element referenced by JDRef. The target of the link is the element's main documentation.

The second static section is activated when JDRef's type is url. It is similar to the first one. Its formula control hyperlink is a URL link and its Expression for URL is: `getJDRefURL(getDGVariable("curPropertyInstance"))`

The third static section is activated for JDRefs of text type. It differs from the previous two in that its formula control has no hyperlink definition. Actually, it can be combined with the first static sections. Since such a JDRef does not refer to any element, the function `getJDRefElement()` always returns null, producing no hyperlinks.

Figure 53 HyperLink tab for a Formula Control



Chapter

15

Documentation Generator and Template Designer Reference

When using the Documentation Template Designer to develop custom documentation templates for the Documentation Generator module, you can reference the internal variables and functions to specify formula expressions and provide flow of control among sections. This chapter lists the variables and functions available for those expressions.

The contents of this chapter are:

- "Documentation generator variables" on page 227.
- "Documentation generator functions" on page 230.

Documentation generator variables

When Together's Documentation Generator executes a template and generates a project report, it uses internal information that includes such things as the project name and the current date/time as well as temporary processing data such as the current model element. DG variables enable you to insert this information into the report. Each variable represents some kind of internal Documentation Generator information available at any particular moment during the generation process.

Not all DG variables are accessible at any instant. Most of them appear only in special areas or inside special sections. Some of them rely on the metamodel types, which are discussed in "Metamodel types" on page 190.

There are three different types of DG variables:

- `String`
- `RwiElement`

- `RwiProperty`

You can access DG variables by using appropriate DG functions in formula expressions:

- `getDGVariable`
- `getDGRwiElement`
- `getDGRwiProperty`

DG variables and properties are case sensitive, however, DG functions are not case sensitive.

For example:

- `getDGVariable("curItemNo")` -- correct usage of `curItemNo` DG variable.
- `getDGVariable("CURItemNo")`-- incorrect usage of `curItemNo` DG variable.

Table 25 lists the DG variables. Table 26 lists the DG functions.

Table 25 Documentation generator variables

Name and type	Description
`curItemNo`: `String`	Current iteration item number (starting at 1). **Availability**: inside property and element iterators **Access via**: `getDGVariable`
`curPropertyName`: `String`	Name of the current property. **Availability**: inside property iterators **Access via**: `getDGVariable`
`curPropertyFullName`: `String`	Full name of the current property as specified in MetaModel file. (See "Metamodel types" on page 190.) **Availability**: inside property iterators **Access via**: `getDGVariable`
`curPropertyType`: `String`	Type of the element property as specified in the MetaModel file. (See "Metamodel types" on page 190.) **Availability**: inside property iterators **Access via**: `getDGVariable`
`curPropertyValue`: `String`	Value of the current property. **Availability**: inside property iterators **Access via**: `getDGVariable`

Table 25 Documentation generator variables (continued)

Name and type	Description

curPropertyInstance: RwiProperty	RwiProperty object of the current property instance. This variable is useful when you need a subproperty of the current property instance.
	Example: Suppose the current model element is a *class* and you need to list information about all *interfaces* implemented by this class. You must create a section that iterates by instances of the IMPLEMENTS property of the current *class* element. Within this iteration section, you can use curPropertyInstance to access the subproperty REFERENCED_ELEMENT, which gives access to information about the implementing class. If you need the *full names* of the implemented interfaces, you can use the expression:
	`findElement(getDGRwiProperty("curPropertyInstance")->`
	`getSubproperty("$referencedElement"))->`
	`getProperty("$fullName")`
	Availability: inside property iterators while iterating by instances of the specified property
	Access via: getDGRwiProperty

curPropertyInstance: String	Value of the current property instance. **Availability**: inside property iterators while iterating by instances of the specified property **Access via**: getDGVariable
curElement: RwiElement	Current model element. **Availability**: inside element iterators **Access via**: getDGRwiProperty
prevElement: RwiElement	Previous element in the current iteration scope. **Possible value**: null if this is the beginning of the scope **Availability**: inside element iterators **Access via**: getDGRwiProperty
diagramMapElement: RwiElement	Diagram element; used to create images maps for diagrams. **Availability**: inside image controls **Access via**: getDGRwiElement
projectName: String	Project name. **Availability**: inside report / page header / footers **Access via**: getDGVariable
nowDateTime: String	Current date/time. **Availability**: inside report / page header / footers **Access via**: getDGVariable
outputFormat: String	Output format of the generated documentation. Use this variable to control the behavior of your templates based on the output format for the generator. **Possible Values**: "RTF", "HTML", "TXT" **Availability**: Anywhere **Access via**: getDGOption

Table 25 Documentation generator variables (continued)

Name and type	Description
reportScope: `String`	Shows the report scope. **Possible values**: `"all_model"` - scope is the whole model `"current_package"` - scope is the current package only `"current_package_recursive"` - scope is the current package and its subpackages `"current_diagram"` - scope is the current diagram only **Availability**: Anywhere **Access via**: `getDGVariable`
stockParam: `String`	Parameter of the stock section call. **Availability**: inside stock sections **Access via**: `getDGVariable`

Documentation generator functions

Table 26 lists the major functions that can be used in formula expressions and enabling conditions.

Table 26 Documentation generator functions

Function name	Signature and Description
getDGVariable	`String getDGVariable(String variableName)` **Parameter**: name of a variable **Returns**: value of `variableName` or the empty string if there is no such variable defined where the function is called.
getDGRwiElement	`RwiElement getDGRwiElement(String variableName)` **Parameter**: name of a variable **Returns**: `RwiElement` value of the variable of the specified name; `null` if the variable is not defined where the function is called
getDGRwiProperty	`RwiElement getDGRwiProperty(String variableName)` **Parameter**: name of a variable **Returns**: RwiProperty type DG variable of the variable of the specified name; `null` if the variable is not defined where the function is called

Table 26 Documentation generator functions (continued)

Function name	Signature and Description

getDGOption	`String getDGOption(String optionName)`
	Parameter: name of a option
	Returns: value of the option with the specified name; the empty string if no such option is defined where the function is called
	Features: The option can be specified for an object of Report Generator (descendant of the class `..gendoc.docgenerator.Generic.GnrReportGenerator`, for example: class `..gendoc.docgenerator.txt.TXTReportGenerator`) using the method `addReportOption (String optionName,String optionValue)`
	Default values for some options can be defined in the template file. These definitions persists even when the Document Template Designer subsequently modifies the template. However, the method `addReportOption` overwrites the options values.
	Example: default values for the options `inclSubpackages`, `inclDoc`, `DTLAdapter`
	`DEFAULT_OPTIONS={inclSubpackages='yes';inclDoc='yes';`
	`DTLAdapter='com.togethersoft.modules.doorslink.DTLAdapter'}`

getParam	`String getParam(String paramName)` **Parameter**: parameter name **Returns**: value of specified template parameter. **Warning**: The requested parameter should be declared in the Template Parameters tab of Template Properties. If the parameter is not declared, calling this function will cause an error message and stop the generator. **Since**: Together 5

Table 26 Documentation generator functions (continued)

Function name	Signature and Description

invokeForName	`String invokeForName(String className, String methodName)`
	`String invokeForName(String className, String methodName, String param1)`
	`String invokeForName(String className, String methodName, String param1, String param2)`
	Parameters:
	`className`: fully-qualified name of the user-provided class.
	This class should not be abstract.
	The Documentation Generator creates an instance of `className` and calls the method `methodName` with this instance. An instance object is created for each entry of `invokeForName` call within each particular expression of template where this function is used. However, the instance is created only during the first call from such an entry and will be used for the next calls, unless the actual `className` parameter is changed.
	`methodName`: name of the method in the class `className`.
	The method should have the following signature:
	`String methodName (..gendoc.api.GenDocContext)`
	The last parameter is an instance of `..gendoc.api.GenDocContext`, which provides the following methods:
	• `RwiReference getRwiReference`
	returns `RwiReference` if the current DG iteration element is an RWI-reference within a diagram. Otherwise the method returns `null`.
	• `RwiElement getRwiElement`
	returns an `RwiElement`. If the current DG iteration element is an RWI-reference, then returned element is `rwiReference.getElement()`. Otherwise, returned element is the current DG iteration RWI-element.
	• `String getParameter1()`
	returns the value of the first optional parameter passed to `invokeForName` function, or `null` if the parameter is omitted.
	• `String getParameter2()`
	returns the value of the second optional parameter passed to `invokeForName` function, or `null` if the parameter is omitted.
	Returns: value calculated by the method named `methodName` of the user-provided class.

Table 26 Documentation generator functions (continued)

Function name	Signature and Description
getContainingDiagram	`RwiDiagram getContainingDiagram()` **Returns***:* RWI-diagram containing the primary reference to the current element **Example**:`rwiElement -> getContainingDiagram()`

isDiagram	`boolean isDiagram()`
	Returns: `true` if the current RWI-element is a diagram.; `false` otherwise. (You can call this function to test any RWI-element of an expression.)
	Example calls:
	`rwiElement->isDiagram()`
	`getDGRWIElement("diagramMapElement")->isDiagram()`
	This function is useful in designing Multi-Frame documentation when you need some special behavior when clicking hyperlinks. For example, you may want clicking a hyperlink to a diagram to reload one frame with a document describing the diagram and another frame with the graphic chart of this diagram. Alternatively, clicking hyperlink to any other model element would load only the document frame. See also , "Creating compound link references" on page 223.

isImported	`boolean isImported()` **Returns**: `true` if the current element in the diagram is a shortcut; `false` if it is not a shortcut
getSubproperty	`String getSubproperty(RwiProperty rwiProperty, String subpropertyName)` **Parameters**: `rwiProperty`: the element property `subpropertyName`: the name of its subproperty **Returns***:* the value of `subpropertyName` contained in `rwiProperty`. **Example call**: `rwiProperty->getSubproperty(subpropertyName)`

Table 26 Documentation generator functions (continued)

Function name	Signature and Description

hasSubproperty	`String hasSubproperty(RwiProperty rwiProperty, String subpropertyName)`
	Parameters:
	`rwiProperty`: the element property
	`subpropertyName`: the name of its subproperty
	Returns: `true` if `rwiProperty` has the specified subproperty; false otherwise.
	Example call: `rwiProperty->hasSubproperty(subpropertyName)String getJDRefType(String jdref)`
	Parameter: JavaDoc reference
	Returns: type of `jdref` as follows:
	`"element"` if `jdref` is a model element (if `jdref` has the form `package.class#member label`)
	`"url"` if `jdref` is a URL (if `jdref` has the form `<a href="URL#value">label</a>`)
	`"text"` if `jdref` has the form `"string"`
	Since: Together 5

getJDRefType	`String getJDRefType(String jdref)`
	Parameter: JavaDoc reference
	Returns: the JavaDoc reference type as:
	`"element"` if `jdref` references a model element (if it has the form: `package.class#member label`, where `package.class#member` represents some model element)
	`"url"` if `jdref` references a url (if it has the form `<a href="URL#value">label</a>`)
	`"text"` if `jdref` has the form `"string"`
	Since: Together 5

getJDRefDisplayName	`String getJDRefDisplayName(String jdref)` **Parameter**: JavaDoc reference **Returns**: text to be displayed in place of the specified JavaDoc Reference: if `jdref` is an `"element"` reference (if it has the form: `package.class#member label`, where `package.class#member` represents some model element) the returned text is `label`. If the `label` is omitted, returns the name of the referenced element. if `jdref` is a `"url"` reference (if it has the form `<a href="URL#value">label</a>`) the returned text is `label`. if `jdref` has the form `"string"`, the returned text is `string`. **Since**: Together 5

Table 26 Documentation generator functions (continued)

Function name	Signature and Description
getJDRefElement	`RwiElement getJDRefElement(String jdref)` **Parameter**: JavaDoc reference **Returns**: If the specified JavaDoc Reference is an "`element`" reference (if it has the form `package.class#member label`, where `package.class#member` represents some model element) and referenced element exists in the model, the function returns this element; otherwise returns `null`. **Since**: Together 5
getJDRefURL	`String getJDRefURL(String jdref)` **Returns**: If the specified JavaDoc Reference is a "`url`" reference (if it has the form `<a href="URL#value">label</a>`) the function returns the text `URL#value`; otherwise, returns an empty string **Since**: Together 5
getCodeElement	`Object getCodeElement(RwiElement rwiElement)` **Returns**: An object. **Details**: The function passes the call to `rwiElement.getCodeElement()` method declared in `com.togethersoft.openapi.rwi.RwiElement` interface. **Usage**: This function is used in the template expressions in conjunction with one of the following functions: `findMember()`, `findNode()`, `findLink()`, `findPackage()`. **Since**: Together 5

Table 26 Documentation generator functions (continued)

Function name	Signature and Description

getCodeElements	`Enumeration getCodeElements(RwiElement rwiElement)`
	Parameter: an RWI-element.
	Returns: code elements associated with the parameter.
	Details: This function passes the call to the `rwiElement.getCodeElements()` method declared in `com.togethersoft.openapi.rwi.RwiElement` interface.
	Usage: This function is used in the template expressions with one of the following: `findDocumentedMember()`, `findDocumentedNode()`, `findDocumentedLink()`, `findDocumentedPackage()`. These functions are helpful for creating hyperlinks from elements on a diagram.
	Example: When `Recognize JavaBean/C++ properties` is on, each JavaBean/C++ property is represented by a single element on a class diagram. Each property actually consists of its attribute plus its setter and getter methods. When you generate the class documentation, those 3 elements will be documented (or at least documentation for accessor methods if private members are skipped). The corresponding element on the diagram associates with an RWI-element that you can obtain via the variable `diagramMapElement` (see also: "Image mapping diagram elements" on page 222). But this RWI-element is actually a kind of a proxy. It is not identical to any of those 3 elements making up the JavaBean/C++ property.
	Thus, you cannot directly use the RWI-element representing a JavaBean/C++ property on the diagram to establish a hyperlink to anything contained in the generated documentation. The solution is to use this expression:
	`findDocumentedMember(getCodeElements(getDGRwiElement("diagramMapElement")))`
	The function `findDocumentedMember()` returns one of the RWI-elements associated with the JavaBean/C++ property and which is definitely presented in the generated documentation.
	Since a diagram contains ordinary elements as well as properties, the expression for diagram hyperlinks connecting RWI-elements may be more complicated:
	`if(getDGRWIElement("diagramMapElement")-> hasPropertyValue("$shapeType","BeanProperty"), findDocumentedMember(getCodeElements(getDGRWIElement("diagramMapElement"))), getDGRWIElement("diagramMapElement"))`
	Since: Together 5

findElement	`RwiElement findElement(String uniqueName)` **Parameter**: string with the unique name of an RWI-element to be found **Returns**: an element of the given name. **Details**: This function passes the call to the `RwiModel.findElement()` method to do the search.

Table 26 Documentation generator functions (continued)

Function name	Signature and Description
findMember	`RwiElement findMember(Object codeElement)` **Parameter**: code element of interest. **Returns**: the member for `codeElement`. **Details**: This function passes the call to the method `com.togethersoft.openapi.rwi.RwiModel.findMember()`. This function should be used together with the function `getCodeElement()`. **Since**: Together 5
findNode	`RwiElement findNode(Object codeElement)` **Parameter**: code element of interest. **Returns**: the node for `codeElement`. **Details**: This function passes the call to the method `com.togethersoft.openapi.rwi.RwiModel.findNode()`. This function should be used together with function `getCodeElement()`. **Since**: Together 5
findLink	`RwiElement findLink(Object codeElement)` **Parameter**: code element of interest. **Returns**: the link for `codeElement`. **Details**: This function passes the call to the method `com.togethersoft.openapi.rwi.RwiModel.findLink()`. This function should be used together with function `getCodeElement()`. **Since**: Together 5
findPackage	`RwiElement findPackage(Object codeElement)` **Parameter**: code element of interest. **Returns**: the package for `codeElement`. **Details**: This function passes the call to the method `com.togethersoft.openapi.rwi.RwiModel.findPackage()`. This function should be used together with the function `getCodeElement()`. **Since**: Together 5

Table 26 Documentation generator functions (continued)

Function name	Signature and Description

findDocumentedMember	`RwiElement findDocumentedMember(Enumeration codeElements)`
	`RwiElement findDocumentedMember(Enumeration codeElements, String subjectSelector)`
	Parameters*:* Code elements and subject selectors to search.
	Returns: found `RwiElement` that satisfies these conditions:
	• it is associated with the passed `codeElements`
	• it is an `RwiMember`
	• it will be presented in all generated documents by its Main Documentation or, if subjectSelector is specified, by its "specific" documentation associated with the passed subjectSelector.
	Returns `null` if the requested element doesn't exist in the model.
	Details: This function should be used together with the function `getCodeElements()`. It calls the method `com.togethersoft.openapi.rwi.RwiModel.findMember()`.
	Since: Together 5

findDocumentedNode	`RwiElement findDocumentedNode(Enumeration codeElements)`
	`RwiElement findDocumentedNode(Enumeration codeElements, String subjectSelector)`
	Parameters*:* See `findDocumentedMember`.
	Returns: See `findDocumentedMember`.
	Details: The function calls the method `com.togethersoft.openapi.rwi.RwiModel.findNode()`.
	Since*:* Together 5

findDocumentedLink	`RwiElement findDocumentedLink(Enumeration codeElements)`
	`RwiElement findDocumentedLink(Enumeration codeElements, String subjectSelector)`
	Parameters*:* See `findDocumentedMember`.
	Returns: See `findDocumentedMember`.
	Details: The function calls the method `com.togethersoft.openapi.rwi.RwiModel.findLink()`.
	Since*:* Together 5

findDocumentedPackage	`RwiElement findDocumentedPackage(Enumeration codeElements)`
	`RwiElement findDocumentedPackage(Enumeration codeElements, String subjectSelector)`
	Parameters*:* See `findDocumentedMember`.
	Returns: See `findDocumentedMember`.
	Details: The function calls the method `com.togethersoft.openapi.rwi.RwiModel.findPackage()`.
	Since*:* Together 5

Table 26 Documentation generator functions (continued)

Function name	Signature and Description
findDocBySubjectSelector	`String findDocBySubjectSelector(String subjectSelectorList)` **Parameter**: list of subject selectors separated with semicolons. **Returns**: the path of the first generated document that contains an area marked with one of the specified subject selectors from the list. The path is relative to the documentation's root directory. Subdirectories are delimited with a slash, `"/"`. If no document is found, the function returns an empty string. **Details**: The function takes the first passed subject selector from the list and checks if there are any generated documents that contain areas marked with this subject selector. If such documents exist, the function returns the one that has been generated first. Otherwise, it iterates to the next subject selector from the list and repeats examination. If all subject selectors are passed and no is document found, the function returns an empty string. **Note**: A blank subject selector is allowed; it will refer to the Main Documentation of an element. **Example**: `findDocBySubjectSelector("package-summary;summary")` returns the first generated document for one of the subject selectors: `"package-summary"`, `"summary"` **Warning***:* This function can be used only inside the Source File Name Expression of the node in FrameSet Structure definition. (See "Defining the frameset structure" on page 216.) **Since***:* Together 5
findDocByTemplate	`String findDocByTemplate(String templateList)` **Parameter**: list of template names (without file name extensions) separated with semicolons. **Returns**: the path to the first generated document produced by one of the specified templates. The path is relative to the documentation's root directory. Subdirectories are delimited with a slash, `"/"`. If no document is found, the function returns an empty string. **Details**: The function takes the first passed template name and checks if there are documents generated by this template. If such documents exist, it returns the one which has been generated first. Otherwise, it iterates to the next template from the passed list and repeats examination. When all templates are passed and no document has been found, the function returns an empty string. **Example**: `findDocByTemplate("all-classes;all-diagrams")` returns the fist document produced by one of the templates: `"all-classes.tpl"` and `"all-diagrams.tpl"` **Warning***:* This function can be used only inside the Source File Name Expression of the node in FrameSet Structure definition. (See "Defining the frameset structure" on page 216.) **Since***:* Together 5

Table 26 Documentation generator functions (continued)

Function name	Signature and Description

checkStockSectionOutput	`boolean checkStockSectionOutput(String stockSectionName, RwiElement rwiElement)`
	Parameters: name of a stock section and an RWI element passed to the stock section as the current model element.
	`stockSectionName` - name of the Stock Section to be tested.
	`rwiElement` - RWI-element passed to the Stock Section as the current model element.
	Returns: `true` if the stock section with the given will produce a non-empty output when it is invoked from a stock section call with `rwiElement` passed to it as the current model element; `false` otherwise. No actual output is produced from a call to this function.
	Warning: If no Stock Section with the specified name is found in the template, the function call issues an error message and stop the generator.
	Example:
	`checkStockSectionOutput("Included Diagram List", getDGRwiElement("curElement"))`
	Since: Together 5

getPropertyExt	`String getPropertyExt(String propertyName)`
	Parameter: name of a property
	Returns: value of the property or empty string if the element has no such property. This function gets any element property available in the Document Generator for the metatype to which this element belongs. It includes the properties provided by RWI and the properties calculated only by the Document Generator . (Names of such properties start with %. See "Metamodel types" on page 190.)
	Example:
	`rwiElement->getPropertyExt(propertyName)`
	`rwiElement` is the element whose property is to be checked.
	See also: `getProperty()`

	Utility String Functions

substring	`String substring(String str, int beginIndex)`
	`String substring(String str, int beginIndex, int endIndex)`
	Parameters: identical to those in the standard Java `String.substring()` methods.
	Returns: a new string that is a substring of `str`.

Table 26 Documentation generator functions (continued)

Function name	Signature and Description

replace	`String replace(String str, String oldStr, String newStr)`
	Parameters: three strings.
	Returns: a new string produced by replacing all occurrences of `oldStr` in `str` with `newStr`. This operation is case sensitive.
	Example: `replace("aBC-BC-bc", "BC", "XY")` returns `"aXY-XY-bc"`
	Usage: This function is especially helpful for a Call to Template section in which the location of the document generated by the called template must be derived from some properties of the current model element (for example, from the full nume of the package where the current element belongs). In this case you can write in the field "Output Directory Expression" an expression such as this:
	`replace(getContainingPackage()->getProperty("$fullName"), "." , "/")`
	See also: "Creating hypertext links (advanced)" on page 221.
	Since: Together 5

duplicate	`String duplicate(String str, int num)` **Parameters**: a string and a non-negative number. **Returns**: a new string resulting from duplicating `str` `num` times. If `num` is 0, returns an empty string. **Example**: `duplicate("aBC", 3)` returns `"aBCaBCaBC"` **Since**: Together 5
length	`int length(String str)` **Parameter**: a string. **Returns**: the number of characters in the string (string length).
str	`String str(Numeric N)` **Parameter**: a number **Returns**: a string representation of the number.
val	`Numeric val(String str)` **Parameter**: a string. **Returns**: the number represented by the string. If conversion is impossible, the function returns 0.
	The following functions, commonly provided in Together formula queries, are also very useful in DG expressions.

getProperty	`String getProperty(String rwiPropertyName)`
	Parameter: the name of an RWI-property.
	Returns: the value of the specified RWI property of the (current) element, or the empty string if the element has no such property.
	Example call:
	`rwiElement->getProperty(rwiPropertyName)`
	`.rwiElement` is the element whose property is returned.
	See also: `getPropertyExt()`

Table 26 Documentation generator functions (continued)

Function name	Signature and Description

hasProperty	`boolean hasProperty(String rwiPropertyName)`
	Parameter: the name of an RWI-property.
	Returns: `true` if the (current) element has the specified property; `false` otherwise.
	Example call:
	`rwiElement->hasProperty(rwiPropertyName)`
	`rwiElement` is the element whose property is to be checked.

hasPropertyValue	`boolean hasPropertyValue (String rwiPropertyName, String value)`
	Parameters: an RWI-property and a possible RWI property value.
	Returns: `true` if the (current) element has specified property with the required value; `false` otherwise.
	Example call:
	`rwiElement->hasPropertyValue (rwiPropertyName, value)`
	`rwiElement` is the element whose property is to be checked.

if	`type if(boolean condition, type value1, type value2)` **Parameters**: any Boolean condition and two values of the same type. The type can be any data type allowed in queries. **Returns**: If the Boolean condition is `true`, the function returns `value1`. If the condition is `false`, the function returns `value2`.

getContainingNode	`RwiNode getContainingNode()`
	Returns: the `RwiNode` element that contains the current element. Can be called for RWI member or node current element.
	Possible call: `rwiElement->getContainingNode()`
	Example: The following expression calculates visibility modifier for the class/interface member:
	`if (hasProperty("$private"), "private",`
	`  if (hasProperty("$protected"), "protected",`
	`   if (hasProperty("$public") && !getContainingNode()->`
	`  hasProperty("$interface"),"public", "")))`
	In this case, the public modifier is printed onlyþ when the containing node is not an interface, since all interface members are implicitly public.

Part

5

CONTEMPORARY SOFTWARE PRACTICES

- Chapter 16, “Refactoring”
- Chapter 17, “Templates and Patterns”
- Chapter 18, “Audits and Metrics”

Chapter

16

Refactoring

Refactoring means rewriting existing source code with the intent of improving its design rather than changing its external behavior. The focus of refactoring is on the structure of the source code, changing the design to make the code easier to understand, maintain, and modify. The primary resource book on refactoring is *Refactoring --Improving the Design of Existing Code* by Martin Fowler (Addison - Wesley, 1999).

Together provides extensive support for refactoring Java projects. Some of the refactoring operations can be applied to other languages as well. This chapter discusses all available refactoring operations. Chapter topics are:

- "Enabling Refactoring" on page 245.
- "Showing code structure" on page 246.
- "Moving classes, interfaces, attributes, and operations" on page 247.
- "Renaming" on page 250.
- "Encapsulating attributes" on page 251.
- "Extracting interfaces and superclasses" on page 252.
- "Summary of refactoring commands" on page 253.

Enabling Refactoring

Refactoring is an activatable feature in Together. You can activate or deactivate such features from Together's main menu.

To activate the Refactoring feature:

1. Select **Tools** | **Activate/Deactivate Features** from the main menu.

2. Go to the Together **Features** tab in the resulting dialog box.
3. Check **Refactoring**.
4. Click **Ok** to activate the checked features and close the dialog box.

When Refactoring is an active feature, you can refactor code in several ways:

- Select **Tools | Refactoring** from Together's main menu.
- Select **Refactoring** from the right-click menu of the currently selected item in the Explorer or the Designer.

The *Refactoring* command has a submenu that lists the individual refactoring commands. The items on the list vary according to the kind of the selected element. The Tools | Refactoring menu contains the full list of refactorings. Only those refactoring that are appropriate for the currently selected element are enabled.

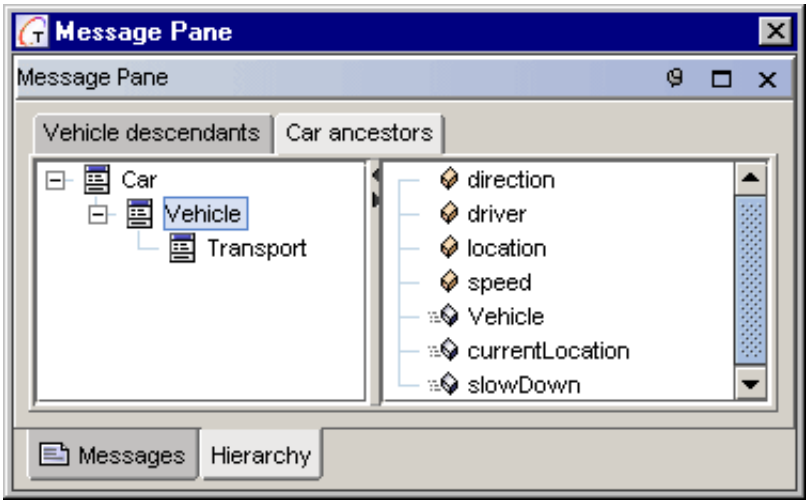
Showing code structure

Together has refactoring commands that enable you to view the part of the class hierarchy that relates to a single class or operation. These "show" commands are available in all languages.They are not refactorings as such, since they do not change the code. However, they can help you investigate the code to determine which refactorings are appropriate.

There are four refactoring commands that show some of the class hierarchy.

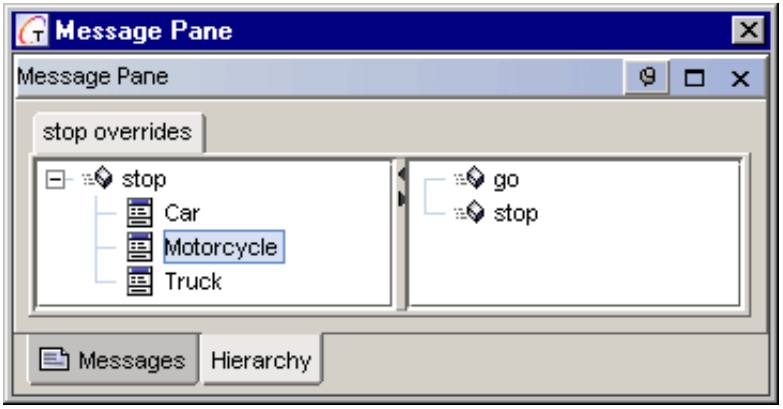
- *Show Ancestors:* (classes and interfaces) shows all of the ancestors of the selected class or interface.
- ***Show Descendants:*** (classes and interfaces) shows all of the descendants of the selected class or interface.
- *Show Implementing Classes:* (interfaces only) shows the classes that implement the selected interface. Shows also all derived interfaces and the classes that implement them.
- *Show Overrides*: (operations only) lists all subclasses of the given class that override the selected operation. Subclasses that do not override this operation are grayed.

Figure 54 The result of *Show Ancestors* of `Car`. The members of the selected class in the left frame (`Vehicle`) are shown in the right frame.



Together displays the results of a show command in a Hierarchy tab in its Message pane. The tab splits into two frames. The left frame displays the tree structure of the relevant classes, with the selected class or operation at the top. The right frame displays the members of the class on the left that you select. You can move the bar dividing the two frames to resize them.

Figure 55 The result of *Show Overrides* of the operation named `stop`.



Moving classes, interfaces, attributes, and operations

You can move classes and interfaces to different packages. You can also move attributes and operations up or down in the class hierarchy.

Moving classes and interfaces

There are two refactoring commands for moving classes and interfaces.

- *Move Class:* moves a class into a different package.
- *Move Interface:* same as *Move Class*, only the move is applied to an interface rather than a class.

To move a class:

1. Apply the move command to the class by selecting the class on the diagram and then doing one of the following:
   - Choose **Refactoring | Move Class** from the right-click menu of the class, *or*
   - Choose **Tools | Refactoring | Move Class** from the main menu
2. In the resulting dialog box, select the target package. Then click **Next** to review the code or **Finish** to complete the move.
3. If you clicked Next on the previous step, review the code and click **Finish**.

If the selected package already contains a class (or interface) of the same name, the **Finish** button at the bottom of the display is disabled. Clicking the **Next** button shows the error messages. You can navigate back to select a different package by clicking the **Previous** button. Clicking the **Cancel** button cancels the attempted move.

With both *Move Class* and *Move Interface*, Together makes the appropriate changes to all of the project code that uses the class or interface to reflect the new package.

Moving attributes and operations in the class hierarchy

There are four refactoring commands for moving attributes and operations.

- *Push Down Operation:* copies an operation from a superclass to a subclass, deleting the original and optionally changing its visibility. If there are no subclasses, Together displays a warning message.
- *Push Down Attribute:* same as *Push Down Operation*, except the move is applied to an attribute rather than an operation.
- *Pull Up Operation:* copies an operation from a subclass to a superclass, deleting the original and possibly changing its visibility.
- *Pull Up Attribute:* same as *Pull Up Operation*, except the move is applied to an attribute rather than an operation.

Tip You can move multiple attributes or operations by selecting several at once in the Designer or Explorer and invoking the right-click menu of the entire selection. If the selection contains both operations and attributes, choose Refactoring | Pull Up Member or Refactoring | Push Down Member.

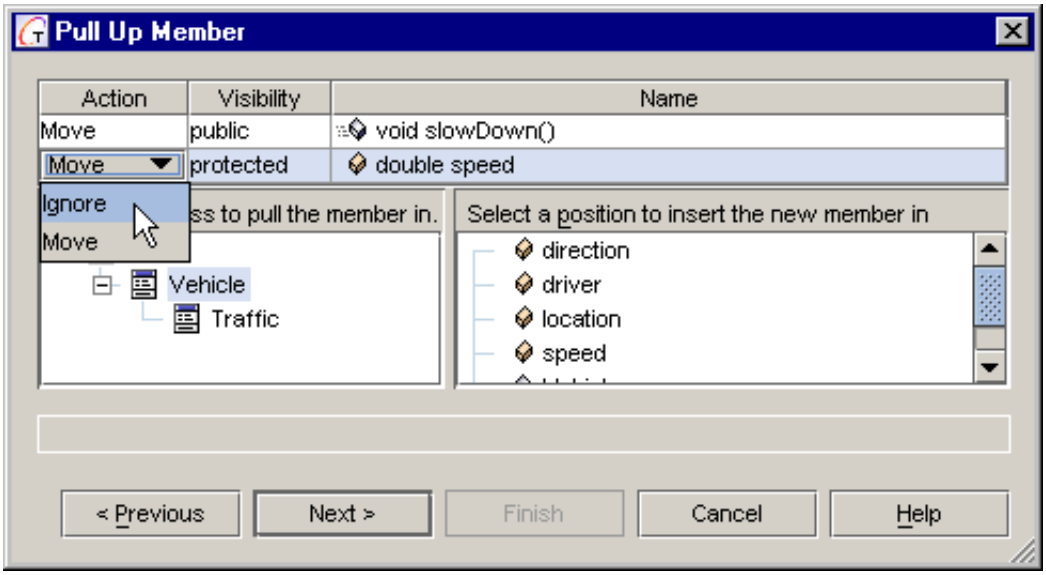
Moving attributes and operations is more complicated than moving classes among packages because class members often contain references to each other. Together issues a warning when a *Pull Up Operation* or *Push Down Operation* has the potential for corrupting the syntax if the operation being moved references other class members. You can elect to move the referenced class members as well, or you can ignore them and correct the resulting code manually.

When you pull up or push down members, Together displays a dialog box for specifying additional information that it must know to make the move. The dialog box takes you through a series of pages:

1. Warning page (optional): For moves that could generate syntax errors (such as pushing down an attribute that is referred to elsewhere in a superclass).
2. Class page: For selecting the target classes that are to receive the members to be moved. Together includes referenced class members in the list of members to move. You can change any of the following options:
   - Action for each member listed (Ignore or Move).
   - Visibility for the moved member (private, public, protected, packageLocal).
   - The target classes to receive the moved members (on the left frame).
   - Where among the target classes' members to make the insertion.

   Figure 56 shows the display to Pull Up an operation named `slowDown`. Since the operation uses the attribute named `speed`, Together gives the option of moving `speed` as well.
3. Code page: Shows changes resulting from the move. The left frame has the tree of classes whose code will change with the move. When you select a class on the left, the right frame displays the original class code and the changes in the code resulting from the move.
4. Error/warning page (optional): Shows a list of the kinds of potential problems resulting from a move. Together will not move any members that cause errors. Click Cancel to leave the code in its original form or click Previous to change the move options.

Figure 56 *Pull Up Member* dialog box displaying target classes. The operation will be moved but the attribute will be ignored.



Renaming

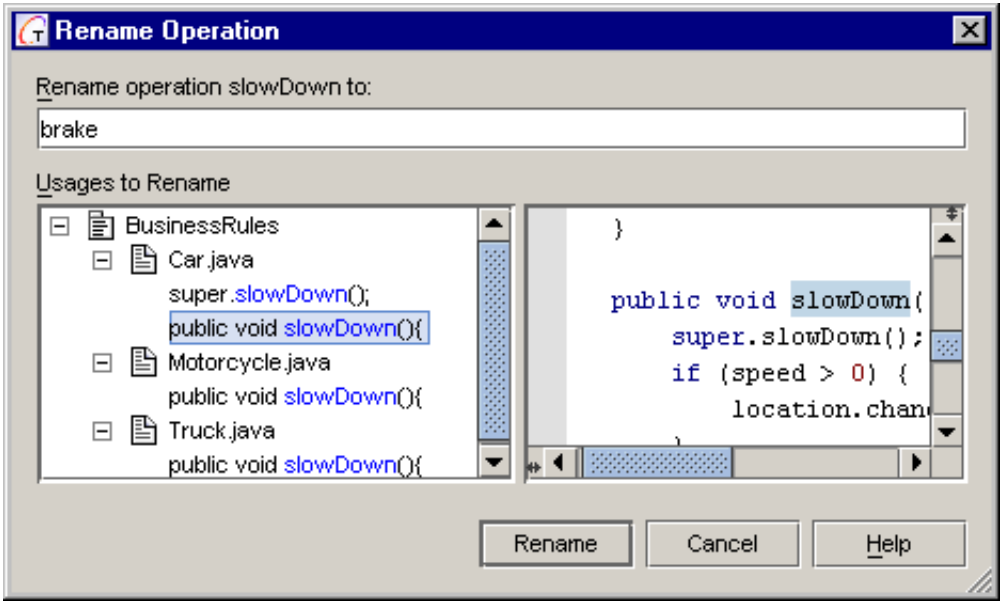
You can rename any code element: class, interface, operation, or attribute. Together propagates the name changes to dependant code in the project files.

To rename a property, use the right-click menu of the property in the Designer pane or the Explorer pane, or use Tools | Refactoring | Rename Property. For any other code element, you can use the Designer, the Explorer, or the Tools menu.

Figure 57 shows the dialog box that results from renaming an operation. At the top is a text field for entering the new operation name. The left frame shows all the tree of usages of the old operation name. Selecting a usage brings up the corresponding code in the right frame. Clicking the Rename button closes the dialog box and makes the name change.

Note You can rename packages simply by changing their names in the Designer or Explorer. Together makes the appropriate changes in this case as well. This action does not require Refactoring to be activated.

Figure 57 Renaming an operation



Encapsulating attributes

Encapsulating an attribute means hiding it by making it private and providing setters and getters where the attribute is used.

You can encapsulate an attribute (or multiple attributes) through its right-click menu by selecting Refactoring | Encapsulate Attribute. The resulting dialog box has multiple displays. With the first display, you can specify:

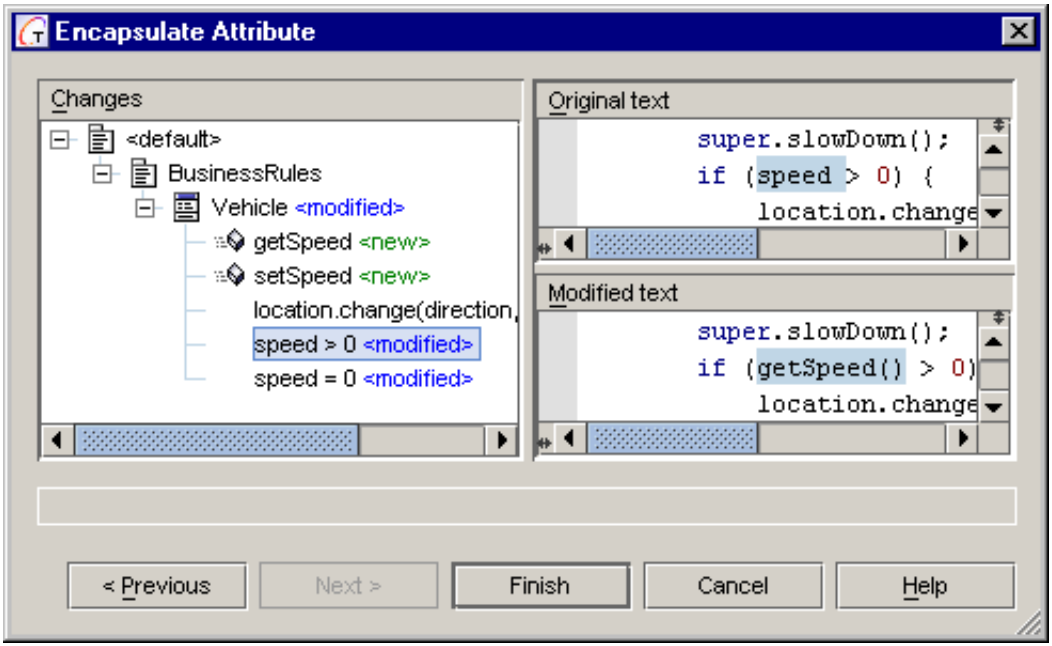
- Visibilities: Dropdown list to select the visibilities of the attribute and its getter and setter methods.
- Names: For getters and setters. You can change the names of either. (There
- Self Encapsulate: Check box for replacing all references to the attribute by calls to its getter and setter methods (even within the class itself).
- Add Comments: Check box for placing new comments into the modified code:
  - JavaDoc comments for the getter and setter
  - Comments for replacing direct assignments to the attribute by calls to the setter.

After selecting the modifications, click the Next button to preview the results in the code. Click Finish to complete the refactoring.

Note Together does not create setters for attributes that are final.

Figure 58 shows the results of encapsulating an attribute named `speed`. The changed lines of code are listed on the left frame. Selecting a line in the left frame displays the original and modified text on the right frame.

Figure 58 Code changes with encapsulating an attribute



Extracting interfaces and superclasses

Together's Refactoring feature enables you to create new interfaces, superclasses, and methods by extracting the information from existing code. You can do extraction in the Designer, the Explorer, and in the Tools menu.

Extracting interfaces and superclasses

Extracting a superclass means creating an ancestor class from several operations of a given class or from several different classes. If any selected class already has a superclass, the new superclass is placed in the inheritance chain between the selected class and its existing superclass.

Extracting an interface means creating a new interface from one or more selected classes. Each selected class would implement the interface.

To extract from multiple classes (or methods), you must select them in the Designer or Explorer and use the Tools menu or the right-click menu of the selection. Choose **Refactoring | Extract Interface** or **Refactoring | Extract Superclass** from the menu.

Tip You cannot extract an interface or superclass from multiple classes unless they have either a static member or an operation with the same signature in common.

The dialog boxes for extracting interfaces or superclasses takes you through a sequence of two pages as follows:

1. Initial page: For specifying the details of the new interface or superclass:
   - *Name:* Enter the name in the textfield at the top.
   - *Package location:* Pick from the display of the project directory.
   - *Select Members:* Determine the action and visibility for the list of effected members. The membership of the list varies according to the kind of element to be refactored. If the element is a class, all the class methods are on the list. If the element is a method (attribute), only the method (attribute) is on the list.
     Clicking the *Action* field for a member displays a dropdown lists with choices of *Ignore, Move, Copy,* and *Abstract.* Clicking the Visibility field for a member displays a dropdown list with choices of *public, private,* and *protected.*

Note Together does not allow you to continue to the next page in the *Extract Interface* or *Extract Superclass* dialog if you fail to enter a name or select a package for the new superclass or interface.

2. Review page: For viewing the relevant code. This page has two frames. The frame on the left lists the code elements changed. The frame on the right splits into an Original text frame and a Modified text frame. Clicking on a code element in the left frame highlights the corresponding code in the frame on the right.

Summary of refactoring commands

Table 27 lists all of the refactoring commands. You can invoke most commands through the Designer, and Explorer, and Editor right-click menus as well as the Tools menu. However, you cannot use the Editor to apply commands to multiple objects..

Table 27 Refactoring commands

Command	Applies To	Comments
Encapsulate Attribute	One or more attributes	
Extract Interface	One class	All class members are listed. Default action is 'Ignore.'

	Several classes	Only members with equal signatures are listed. Default action is 'Abstract.'
	Several members of a single class	Selected members are listed. Default action is 'Abstract.'
Extract SuperClass	One class or interface	All class members are listed. Default action is 'Ignore.'
	Several classes or interfaces	Only members with equal signatures are listed. Default action is 'Move.'
	Several members of a single class or interface	Selected members are listed. Default action is 'Move.'
Move Class	One or more classes	
Pull Up Member	One or more members of a single class	
Pull Down Member	One or more members of a single class	
Rename Attribute	One attribute	
Rename Class	One class	
Rename Interface	One interface	
Rename Operation	One operation	
Rename Property	One property	
Show Ancestors	One or more classes or interfaces	
Show Descendants	One or more classes or interfaces	
Show Implementing Classes	One or more classes or interfaces	
Show Overrides	One operation	

Chapter

17

Templates and Patterns

This chapter provides instructions for the following:

- "Code templates" on page 255.
- "Custom code templates" on page 259.
- "Patterns" on page 265.
- "Using templates and patterns" on page 268.

Code templates

Code templates are used to generate the initial source code and default property values for new modeling elements you create in Together. A simple example is the default template for a class. When you create a new Java class, its default name is "Class1," and the default code generated for it is:

```
public class Class1 {
}
```

By modifying the appropriate template, you can change the name to "New1" (or whatever you want). More importantly, you can also modify the default code, adding default attributes or operations. For example, you could change the default code for a Java class so that a default `addNew()` operation is always generated when you create new classes in diagrams:

```
public class Class1 {
  public void addNew() {
  }
}
```

This may seem trivial at first glance, but consider the implications for things like Enterprise JavaBeans (EJBs). When you use the one-click EJB feature, you get default source code for an entity or session EJB. By modifying the respective code templates, you can customize the default code for EJB classes, home and remote interfaces, and so on, adding fields, properties, business methods, or whatever you want to the default source generated for new instances of the particular element in diagrams.

A class template is stored in a folder whose name corresponds to the name of this template. This folder contains the file `%Name%.*` (with an extension specific to the selected language), and an optional properties file. Link and member templates have the template name with the `*.link` or `*.member` extension, respectively.

Template properties

Properties of the template are defined in an optional file `template_name.properties` in the same subfolder. This file includes values that will be substituted instead of the macros when a new object is generated, a flag that specifies whether this template will be displayed in the Choose Pattern dialog, and other information. Table 28 presents the properties available.

Table 28 Code Template properties

Property	Description
defaultName	The name of the created object, which is used as a starting value. For example, Class1 for the first generated class, Class2 for the next one, and so on.
defaultType	Defines types of attributes and return types of operations.
hideInChooseList	If this property is present in the *.properties file, the template is ignored by the Pattern Chooser panel.
generatePrologueEpilogue	If true, a pre-defined prologue and epilogue will be generated.
pasteClassesToOneFile	Some of the class and interface templates stipulate generation of two classes. If this flag is true, both objects are generated in a single file.
singleOccurrencePerClass	This property refers to operators and members, and specifies that this operator or member can occur in the generated class only once.
patternDescription	Contains a brief description of this template in HTML format.
doNotKeepTag	Contains the tag name that should not be preserved when an object is replaced with another one. For example, `doNotKeepTag=link` means that the link tag should be omitted in a new link.

The textual pattern `Default_Class` is used every time a new class is created. The same is true of the interfaces, associations, aggregations, dependencies, and so on, whose names begin with "Default_." These templates never show up in the Pattern Chooser panel.

Using template macros

You will notice in the default code delivered with Together that several "template macros" are used. For example, the default class code uses the `%Name%` macro:

```
public class %Name% {
}
```

This macro expands to the default class name specified in the properties (Class1 by default). You can use any of the available macros in the code. Template macros are documented in the Templates Appendix of this guide. Each template type handles certain pre-defined macros:

Table 29 Template types and corresponding macros

Template type	Macros handled
Class	`%Name%,%Class_Name%`
Member	`%Name%,%Type%,%Class_Name%`
Link	`%Name%,%Type%,%Dst%`

The macros not supported by a certain template type are ignored by the parser, and a template containing unsupported macros will be inapplicable.

In contrast to pre-defined macros, each template type can handle an unlimited number of user-defined macros.

Default properties in templates

The template node for each modeling element contains two subnodes: the first for the default source code, the second for the default properties.

Each template contains a set of default properties whose values you can modify. For example, the default name is specified in the properties. Thus, to change the default code for an element, open and edit the source code node; to change default values of properties, edit the properties node. (See "Editing code templates" on page 259 below.)

Important Never edit the default templates. Improper modifications can result in incorrect functioning of Together, and make it impossible to create diagram elements.

Browsing available code templates

You can browse through the available code templates using the Directory tab of the Explorer. You do not need an open project.

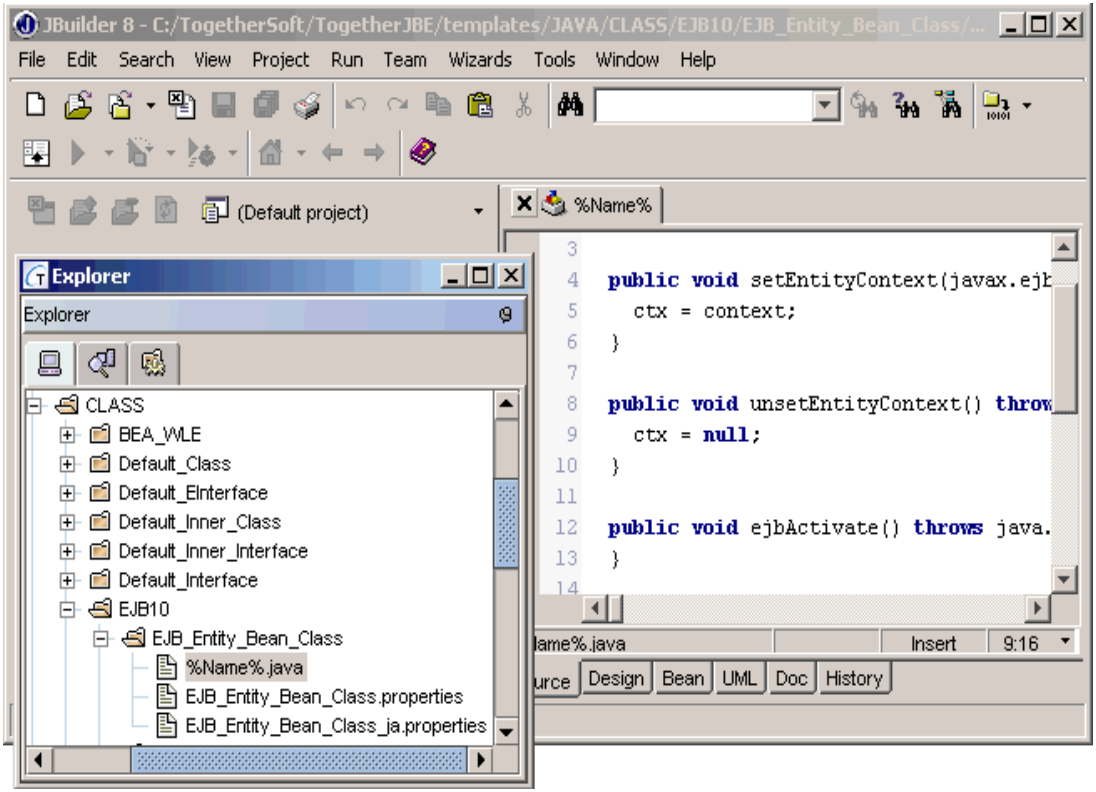
To see the main template categories:

1. Click on the Directory tab of the Explorer.
2. Expand the *Templates* node.
3. Expand the *JAVA* node.

Note that while all templates are available to view and edit, not all templates function in all products. Functionality depends upon your product type.

Each main category has subcategories for class templates, link templates, and member templates. The figure below shows the structure of the *Templates* node on the Directory tab, and the default code for the selected element. This is the code generated for this element (in this case an Entity EJB) whenever a new one is created in the visual model. You can edit this code using the JBuilder Editor (see the next section) to change the default source generated for new instances of the modeling element. For example, for the EJB shown in Figure 59 you could add default declarations for finder or business methods.

Figure 59 Using code templates



Editing code templates

You can edit the default source code specified by a template, the default values of properties, or both using the JBuilder Editor. You can open multiple source templates or properties in the editor at once, thus facilitating clipboard operations between templates. Changes to source code or properties are automatically saved when you leave the Editor. You can manually save changes using the File menu or the main toolbar.

To open one template for editing:

1. Navigate to the template node in the Directory tab of the Explorer and select it.
2. On the right-click menu of the node, choose **Tools | External Editor.**

To open multiple templates for editing:

1. Navigate to a template node in the Directory tab of the Explorer and select it.
2. On the right-click menu of the node, choose **Edit in New Tab**.
3. Repeat steps1 and 2 for additional templates as desired.

Note You can use the currently configured External Editor to edit the code templates. On the right-click menu of the template, choose **Tools | External Editor**.

Applying source formatting

Source code formatting applies to templates in the same way as to other sources. You can configure a number of source code formatting options in the Source Code category of the Options dialog, such as indenting, comment format, treatment of spaces, and so on. Once you have edited a template to its final state, you can apply the currently configured source formatting options to the code.

To apply source formatting options:

1. Select the *source code* node (not the *properties* node) of the template in the Explorer.
2. On the right-click menu of the node, choose **Tools | Format Source**.

Tip The same command is available on the right-click menu of the Editor.

Custom code templates

Using Together you can create your own templates and groups of templates, either using an Expert, or manually. You can also collect appropriate templates into groups, provide group-level descriptions, and customize the way the templates show up in the Pattern Chooser.

When you create custom code templates, keep in mind that there is a rigid dependency between the template type and the contents of the Inspector. For classes and interfaces, a single template-related control with the class name appears in the Inspector. The value of the class name replaces the `%Name%` macro in the template.

For links, the *Name* field refers to the `%Name%` macro, and the *Link destination* field refers to the `%Dst%` macro.

For members (attributes or operations), there is the *Name* field for the `%Name%` macro, and the *Type* field for the `%Type%` macro.

If a certain macro is not included in the template body, the corresponding field does not show up in the Inspector. However, manipulations with the pre-defined macros require special consideration, to avoid producing useless templates.

Working with the Code Template Expert

Together provides a convenient way to create custom code templates using the Code Template Expert. Open the expert from the Tools menu, or from the right-click menu of a diagram object. The expert creates template and properties files according to the rules described above.

In the Code Template Expert you can:

- Define template properties.
- Edit template contents using template macros.
- Delete custom code templates.

To create a new code template:

1. On the main menu, choose **Tools | Code Template Expert**.
2. On the first page of the Code Template Expert, choose the target **Java** language and category of template, and click **Next** to continue.
3. On the second page of the Expert, click **New Template** to add a new template.

Note Click the **New Folder** button to create a group of templates.

4. On the third page of the Expert, specify the template name, and set the options for template properties.
5. Click **Edit Template Code**.
6. In the Edit Code Template window, type in the source code, using the macros buttons. Apply source formatting if necessary, and click **OK** to complete editing.
7. On the third page of the Expert, click **Finish** to complete.

Note The controls of the Edit Code Template dialog are described in the relevant topic of the online help system.

The newly created templates are displayed in the Explorer pane. Now you can use them to create classes, members, and links by patterns. This technique is described in the sections "Creating classes and links by patterns" on page 269 and "Creating members by pattern" on page 270 in this chapter.

To edit an existing code template:

1. On the main menu, choose **Tools | Code Template Expert**.
2. On the first page of the Code Template Expert, choose the category of template to be modified, and click **Next** to continue.
3. On the second page of the Expert, select the desired template and click **Next**.
4. On the third page of the Expert, edit the template name, and set the options for template properties as required.
5. To edit the source code, click **Edit Template Code**.
6. In the Edit Code Template window, modify the source code, using the macros buttons. Apply source formatting if necessary, and click **OK** to complete editing.
7. Click **Finish** to complete.

To delete a code template:

1. On the main menu, choose **Tools | Code Template Expert**.
2. On the first page of the Code Template Expert, choose the category of template to be modified, and click **Next** to continue.
3. On the second page of the Expert, select the desired template and click **Remove Template**.
4. Confirm deletion, and proceed with the other templates or close the expert.

Important The Code Template Expert protects the integrity of Together. Default templates used to create one-click diagram elements, whose names start with "Default," may not be deleted by any means. For this reason, if you select any default template in the Expert tree, the Remove button is disabled. However, it is still possible to modify the default templates.

Creating custom code templates manually

If you so wish, you can modify the code templates by means of editing the relevant files in the Templates folder. In this case you are fully responsible for the correctness of created templates.

To create a custom code template manually:

1. According to the required template type create a folder in the appropriate location. The name of this folder should be the same as the name of the template being created. Spaces are not allowed in the folder name and should be replaced with underscores. Further, when this pattern is displayed in the Pattern

Chooser panel, the underscores in the folder name will be substituted with spaces.

For example: `Templates\Java\CLASS\My_pattern`.

2. In this folder, create the file `%Name%.java` and optional properties file, whose name is the same as the template folder name.
3. Create the source code for the template file.
4. Write the contents of the properties file.

Tip If you want your template to be displayed in the Pattern Chooser panel, do not include the `hideInChooseList=true` property in the properties file. Even if you set this flag to false, the Pattern Chooser panel still ignores the template. This line should be omitted.

The newly created templates show up in the Together Explorer pane when you restart Together. Now you can use the new templates to create classes and links by patterns.

Groups of templates

For more convenience, you can gather relevant patterns into folders and provide general descriptions for the groups of patterns.

To create a folder:

1. In the Code Template Expert, click **New Folder**.
2. In the Create New Folder dialog, enter the desired name.

Further, using the Pattern Chooser, you may want to see group-level descriptions.

To provide a description for a group of patterns, create the file `description.html` under the folder in reference, and write the necessary information. The description is displayed in the Description area of the Pattern Chooser when you restart Together.

Displaying custom template names

As mentioned earlier, templates are stored in folders with appropriate names. However, you might want to see more sensible names in the Pattern Chooser, with spaces, quotes, and apostrophes. To change the display name, add the following line to the template's `*.properties` file:

```
patternDisplayName=new name
```

The new name immediately shows up in the Pattern Chooser, but the actual name is still displayed in the Templates node of the Explorer.

It is also possible to rename template folders in the Pattern Chooser. To do that, create the file `folder_name.properties` in the upper-level directory. This file should contain only one line:

```
patternDisplayName=new name
```

When you restart Together, the new name is displayed in the Pattern Chooser.

Example

Rename the group of templates *Robustness*, which resides in `$TGH$/templates/java/class`, to *Robustness Diagram*.

To rename the group:

1. Create the file `$TGH$/templates/java/class/Robustness.properties`.
2. Add the following line:

   ```
   patternDisplayName=Robustness Diagram
   ```

3. Restart Together.
4. Open the Choose Pattern dialog and observe the new name in the Pattern Chooser panel.

User-defined macros

In addition to the standard macros, an unlimited number of user-defined macros are allowed for all template types. The names of the user-defined macros follow the same syntax rules as the template folder names (spaces not allowed). The Inspector provides a special control for each macro with the name of this macro, and the underscores are replaced with spaces.

Example: The control "Custom Attribute Name" corresponds to the macro `$Custom_Attribute_Name$`.

It is possible to assign default values to the user-defined macros. These values initialize appropriate controls in the Choose Pattern dialog.

To assign a default value to a user-defined macro:

Under the *Templates* node of the Explorer, navigate to the desired template folder.

1. Open the properties file of the template for editing.
2. In the JBuilder Editor, add the line `default.$Custom_Attribute_Name$=<defaultValue>`.
3. Save changes.

The macros without pre-defined default values are initialized with an empty string.

Creating templates with custom file extensions

Together makes it possible to generate files with custom file extensions, using appropriate templates. This requires certain spade-work to be done.

Defining the source for extension of the newly generated files

When a class is generated by template, its extension can be taken from the two possible sources:

- textual templates extension, defined in `$TGH$/config/resource.config`,

  or

- codegen default extension, defined in `$TGH$/config/java.config`

By default Together uses the codegen default extension, but you can make Together use the template extension instead.

To use a textual template extension for the generated file:

1. Open the file `$TGH$/config/<language>java.config` for the appropriate language.
2. In the section Codegen options, uncomment the following line:

```
;codegen.java.use_template_extensions = yes
```

Defining the custom template extension

To define the template extension:

1. Open the file `$TGH$/config/resource.config`.
2. Add the following lines to the appropriate language section:

```
resource.file.NewExtension.extension = "NewExtension"
resource.file.NewExtension.name =  ["filetype/java.source"]
resource.file.NewExtension.type = "source"
resource.file.NewExtension.language = "java"
resource.file.java_source.extension.2 = "NewExtension"
```

Creating files with custom template extensions

When the source of the extension and its text are defined, you can create a new template and generate files with new extensions using this template.

To create a file with custom template extensions:

1. Start Together, and choose **Tools | Code Template Expert** command on the main menu
2. Create a new class template, as described in the section "Custom code templates" on page 259.

3. In the Together installation, find the newly created template file, and change its extension as required:

   For example, change `$TGH$/templates/JAVA/CLASS/<new template>/%Name.java`

   to `TGH$/templates/JAVA/CLASS/<new template>/%Name.NewExtension`

4. In Together, click **Class by Pattern** button on the class diagram toolbar, and choose the new template with custom extension. The class with the specified extension is generated.

Patterns

Together delivers a unique capability to extend its functionality externally by using modules and patterns. In this topic you will learn about the basic concept of patterns, how to use the "pre-fab" patterns delivered with Together, and how to deploy your own patterns. Modules are discussed in Chapter 30, "Together Open API".

What are patterns?

Together patterns are public Java classes that implement the interface `com.togethersoft.openapi.sci.pattern.SciPattern`.

Patterns are intended to:

- Create frequently used elements.
- Modify existing elements.
- Implement useful source code constructions or solutions in your model.

Among the patterns provided with Together, there are the Coad Components, GoF patterns including Visitor, Observer, and Singleton, a set of EJB classes for various specifications, a set of TagLib patterns, and many others.

You can develop your own patterns and reuse them in Together. Patterns are implemented using the SCI API. To develop your own patterns, follow the basic procedure outlined in the section "Creating patterns" on page 266. You will also need to study the documentation for the Together API. (For more information on the API documentation, see Chapter 30, "Together Open API".) You can study examples of mature patterns in the `$TGH$\modules\com\togethersoft\modules\patterns` directory in your installation and you can also obtain the sources of patterns at the Together Community web site:

http://community.togethersoft.com/.

Elements generated by patterns

The package of a pattern and its subpackages define the target elements for the pattern (or elements produced by the pattern).

Table 30 Elements generated by patterns

Element	Package
Class	`com.togethersoft.modules.patterns.JAVA.CLASScom.togethersoft.modules.patterns.UNIVERSAL.CLASS`
Link	`com.togethersoft.modules.patterns.JAVA.LINKcom.togethersoft.modules.patterns.UNIVERSAL.LINK`
Member	`com.togethersoft.modules.patterns.JAVA.MEMBERcom.togethersoft.modules.patterns.UNIVERSAL.MEMBER`

Pattern behavior

Behavior of a pattern is defined by its properties set. The `SciPattern` interface defines the methods that you should implement in your pattern.

Table 31 Methods of the `SciPattern` interface

Method	Description
`apply`	Makes the pattern perform the desired actions.
`canApply`	Checks whether the pattern can be applied to the target object or objects with the current values of the pattern properties.
`prepare`	Checks if it is possible to apply this pattern to the target object or objects at all, and makes some startup preparations for the pattern. It returns `true` if everything is okay, and `false` if the pattern cannot be applied to the target object or objects at all.
`getProperties`	Returns an instance of the `PropertyMap` that contains the set of all the pattern properties.

Other information about pattern-related interfaces and pattern properties can be found in the main API documentation (`/doc/api` folder in your Together installation).

Creating patterns

It is possible to create a pattern hierarchy within the packages described above. In this case Together will use a name of a subpackage as a name for a group of patterns defined in this package.

For example, a pattern defined in the package

```
com.togethersoft.modules.patterns.JAVA.CLASS.foo_Pattern
```

will be displayed in a separate group called “foo Pattern” in the class pattern chooser dialog (Together automatically replaces an underscore sign with a space).

The following example shows how to develop a simple pattern step by step, using Together:

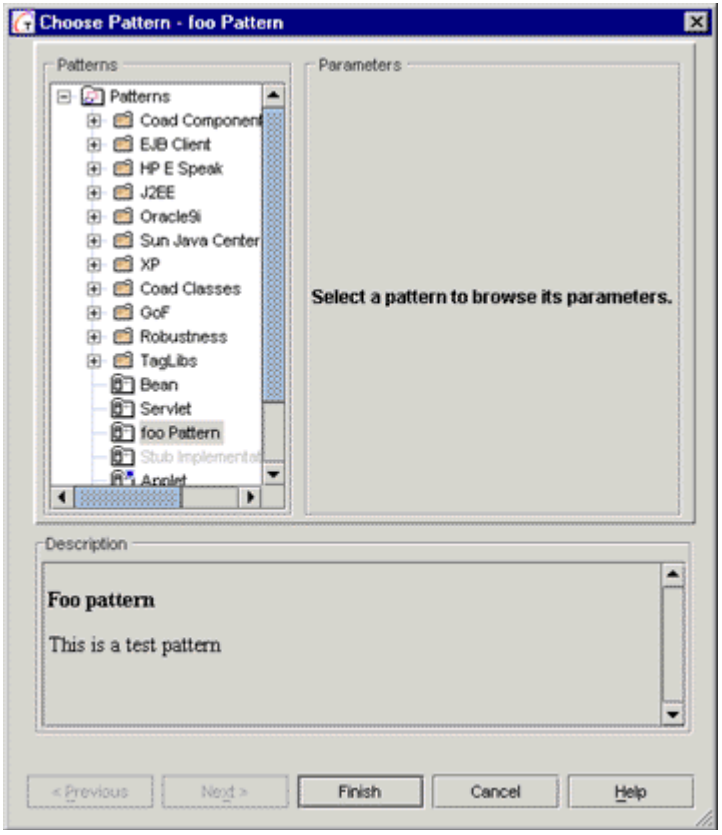
1. Create the folder `foo_Pattern under com.togethersoft.modules.patterns.JAVA.CLASS`. This pattern will work with Java classes only.
2. Create a JBuilder project and name the project `foo_Pattern (for example, myprojects/foo_pattern)`.
3. Add `$TGH$\lib\openapi.jar` and `$TGH$\lib\openapi.jar` as a required library for the project.
4. From the main menu, choose **Project | Open project in** Together. Switch focus to Together.
5. Add a new class to the default diagram, naming it `Pattern.java`.
6. Right click on `Pattern`, and choose **Edit**. Switch focus to JBuilder.
7. Add the following code in the JBuilder Editor:

```
package com.togethersoft.modules.patterns.JAVA.CLASS.foo_Pattern;
import com.togethersoft.openapi.sci.pattern.SciPattern;
import com.togethersoft.openapi.util.propertyMap.PropertyMap;
import com.togethersoft.openapi.util.propertyMap.DefaultPropertyMap;
public class Pattern implements SciPattern {
public boolean prepare () {
return true;
}
public boolean canApply() {
return true;
}
public void apply() {
}
public PropertyMap getProperties(){
return myProperties;
}
private PropertyMap myProperties = new DefaultPropertyMap();
}
```

8. Choose **Project | Project Properties** from the JBuilder main menu. Click the **Build** tab in the resulting dialog box.
9. Select *javac* from the **Compiler** drop down list, and click **Ok**.
10. Choose **Project | Make Project** from the JBuilder main menu.
11. Place the compiled class file in the `$TGH$/modules/com/togethersoft/modules/patterns/JAVA/CLASS/foo_pattern` directory.

The Choose Pattern dialog displays the pattern. If you click the Class by Pattern button, the tree-view of available patterns and templates displays the new pattern as shown in Figure 60.

Figure 60 New pattern in the Choose Pattern dialog



Note that the new pattern is not marked with an asterisk in the patterns tree-view. This denotes the difference between binary patterns and textual templates.

Providing pattern descriptions

To create a description for your pattern, you need to place a file named `Description.html` in the directory with your pattern. The contents of this file are displayed in the Pattern Chooser dialog in the *Description* section.

Using templates and patterns

Though different in nature, both templates and patterns are used to create or refactor diagram elements via the Choose Pattern dialog. Textual templates in the pattern chooser are marked with an asterisk, to distinguish them from the binary patterns. However, both reusable components are applied in the same way.

If a textual template is selected in the tree-view of available patterns, the preview of the code is displayed in the right pane. For binary patterns, the preview is not available. You can find a detailed description of controls in the online help system.

Creating classes and links by patterns

Together makes it easy for you to apply patterns and templates when creating classes or links. To create classes or links during modeling, you can use the buttons in Table 32 on the class diagram toolbar.

Table 32 Toolbar buttons that use templates and patterns

Tool-tip	Icon	Description
Class by Pattern		Creates a new class using selected pattern to define how code is generated.
Link by Pattern		Creates a relationship link using selected pattern to define how code is generated.

Both buttons launch the Choose Pattern dialog, displaying the available patterns for the respective operation. You can access Together predefined patterns, plus any that you implement yourself.

To create a class or link by pattern:

1. Click the appropriate toolbar button.
2. If creating a class, click on the Designer pane to open the Choose Pattern dialog for classes. If creating a link, drag it from the source class to the destination and drop it to open the Choose Pattern dialog for links.
3. Select the pattern you want for the new class or link.
4. If the Next button is enabled, there are properties or parameters for the pattern. Click **Next** to display them. (If you know the pattern and want to accept the defaults, click **Finish**.)
5. Set properties or parameters as desired and click **Finish**.

Note If you are using the Microsoft JVM, note that `java.rmi` classes, required for compilation of the EJB pattern, are not present in this JVM.

Important The result of applying a pattern to a class depends on the way it was invoked. If you are going to create a new class by pattern, make sure you click on the Designer pane, rather than on a class shape. In the latter case, the class where you click is regarded as a container class, and the newly created class by pattern is added as an inner class.

Invoking the Choose Pattern command from the class right-click menu enables refactoring of the class according to the selected pattern.

Creating a stub implementation

When creating an inheritance link between a class and another class or interface, the methods and members are not automatically added to the child class. This problem is solved using the Stub Implementation pattern.
If the destination of a link is an interface, the pattern makes the class-source implement that interface, and creates in a class the stubs for all the methods found in the interface and all its parent interfaces.

If the destination of a link is another class, then this pattern makes the class-source extend the class-destination, and makes stubs for all the constructors found in the class-destination. These constructor stubs simply call the corresponding constructors in the class-destination.

It is also possible to copy Javadoc comments of the inherited methods by checking the *Copy documentation* box in the pattern dialog.

To create an inheritance link with stub implementation:

1. On the diagram toolbar, click the Link by Pattern button.
2. Click on the source class and drag-and-drop the link to the destination class or interface.
3. In the Choose Pattern dialog for links, select *Stub Implementation*.
4. Check the *Stub constructors* and *Copy documentation* boxes, if necessary.
5. Click **Finish** to complete. The inheritance link is created, and the stubs for the inherited methods are generated in the source class.

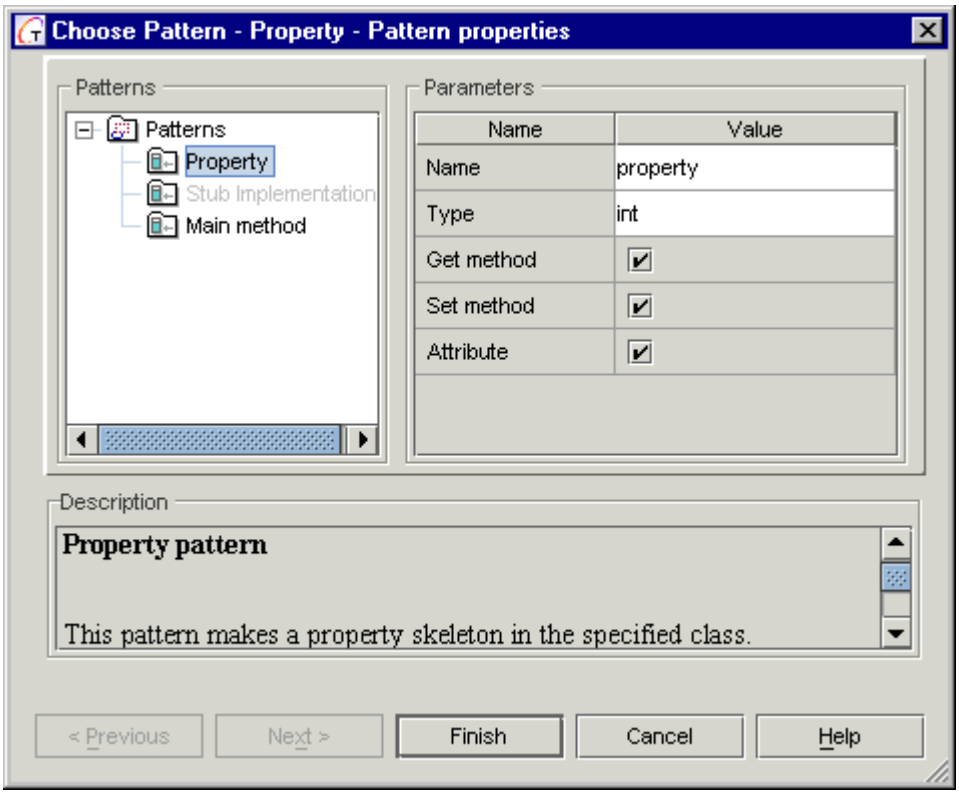
Creating members by pattern

Unlike classes and links, there is no toolbar button for this operation.

To create a member by pattern:

1. Open the class right-click menu.
2. Choose **New | Member by Pattern**.
3. In the Choose Pattern dialog, select the desired property pattern.
4. Enter property name and type, and check the boxes for accessor methods and attributes if necessary.

Figure 61 Choose Pattern dialog for Members



Choosing a pattern for a Member

You can also apply patterns to control the way implementation code is generated for an attribute or operation.

The Choose Pattern dialog displays the *Member* and *Link* nodes. Member patterns can make the selected member an attribute, an operation, or even regenerate the code for the property with `get` and `set` method. Link patterns can make the selected member a link.

To apply a pattern to a member:

1. Select an attribute or operation in a class.
2. On the right-click menu of the member, click **Choose Pattern**. The Choose Pattern dialog opens, displaying the available patterns for members.
3. Select the pattern you want for the member.
4. Click the **Next** button to display the page with the properties or parameters of the pattern, or click **Finish** to accept the defaults. (If there are no properties or parameters, the Next button is disabled.)

5. Set properties or parameters as desired and click **Finish**.

Note The Choose Pattern command is enabled only if the selected element satisfies certain criteria which make it possible to regenerate the code without any losses. Thus, if an operation already has some code in its body, the command is disabled.

Refactoring with patterns

You can also refactor existing classes or links, replacing an existing pattern with a better one. You can perform this singly, or to multiple selected elements.

To refactor classes or links:

1. Select the elements in the Designer pane.
2. Click **Choose Pattern** on the right-click menu of the selected elements to display the appropriate Choose Pattern dialog.
3. As above, select the desired pattern for refactoring, set any properties or parameters, and click **Finish**.

Tip You can also access the Choose Pattern dialog from the right-click menus of class diagram elements in the Explorer.

Chapter

18

Audits and Metrics

This chapter includes the following topics:

- "Overview of audits and metrics analysis" on page 273.
- "Running audits and metrics in Together" on page 275.
- "Audits and metrics supported for Java" on page 288.
- "Running audits and metrics from the command line" on page 292.
- "Extending the QA module" on page 294.
- "Additional information sources" on page 295.

Overview of audits and metrics analysis

Together provides audits and metrics as Quality Assurance features to unobtrusively help you enforce company standards and conventions, capture real metrics, and improve what you do. Although audits and metrics are similar in that they both analyze your code, they serve different purposes.

When you run audits, you select specific rules to which your source code should conform. The results display only the violations of those rules so that you can examine each problem and decide whether to correct the source code or not. Together provides a wide variety of audits to choose from, ranging from design issues to naming conventions, along with descriptions of what each audit looks for and how to fix violations.

Metrics, on the other hand, quantify your code. It is up to you to examine the results and decide whether they are acceptable. Metrics results can highlight parts of code that need to be redesigned, or they can be used for creating reports and for comparing the overall impact of changes in a project. Along with the full set of metrics, Together provides tips for using metrics and interpreting results.

Audits and metrics are run as separate processes. Because the results of these two processes are different in nature, Together provides different features for interpreting and organizing the results. Note that some of the features and procedures described in this chapter apply to both audits and metrics, while some are specific to either one or the other.

How should metrics be used?

The following recommendations are taken from the book *Object-Oriented Metrics: Measures of Complexity* by Brian Henderson-Sellers. For other resources, see the list "Additional information sources" on page 295 at the end of this chapter.

The use of metrics depends on the maturity of the company, and therefore on the level of organization and control of the development process. The set of metrics in use can be juxtaposed to the organizational level, as shown in Table 33.

Table 33 Identification of the organizational level in the Capability Maturity Model, and metrics usage

Level	Characteristic	Metrics to use
1. Initial	Ad hoc	Baseline
2. Repeatable	Process dependent on individuals	Product
3. Defined	Process defined, institutionalized	Product
4. Managed	Measured process (quantitative)	Process + feedback to control
5. Optimizing	Improvement fed back to process	Process + feedback for changing process

Baseline metrics – "no order" in metrics using.
Product metrics – metrics on estimating and measuring size.
Process metrics – metrics on estimating effort (depends on size and team productivity).

However, dividing metrics into two groups (the product and the process) is relative because the same metric might represent product size and also impact productivity.

Metrics can be collected for various purposes such as cost estimation, project management, quality assessment, calculating productivity, detecting defects, or program maintenance. To apply metrics successfully, it is important for managers and developers to observe a number of rules, summarized in Table 34.

Table 34 Rules of etiquette for applying software metrics

Management level	Rules
Functional management	1. Do not allow anyone in your organization to use metrics to measure individuals. 2. Set clear goals and get your staff to help define metrics for success. 3. Understand the data that your people take pride in reporting; do not use it against them; do not ever hint that you might. 4. Do not emphasize one metric to the exclusion of another. 5. Support your people when their reports are backed by data useful to the organization.
Project management	6. Do not try to measure individuals. 7. Gain agreement with your team on the metrics that you will track, and define them in a project plan. 8. Provide regular feedback to the team about the data it helped collect. 9. Know the strategic focus of your organization, and emphasize metrics that support the strategy in your reports.
Project team	10. Do your best to report accurate, timely data. 11. Help your managers to focus project data on improving your process. 12. Do not use metrics to brag about how good you are or you will encourage others to use other data to show the opposite.

Together provides tips on how to use the results of specific metrics. These tips are displayed on the Advice tab for the Distribution graph (see "Distribution graph" on page 284). Remember that when interpreting values for metrics, it is not a good idea to apply the suggested limits dogmatically. For every limit, there are good arguments for exceeding the maximum. However, consider such cases as exceptions, rather than the rule.

Running audits and metrics in Together

When the Quality Assurance feature is activated, the commands Audits, Metrics, and Load Metrics Results are available on the Tools menu.

- To process the entire project, choose Audits or Metrics from the Tools menu.
- To process only specific classes, packages, or diagrams, select the elements either on the model tree in the Explorer or on the diagram, and choose Audits or Metrics from the right-click menu.

You can create, save, and reuse custom sets of audits and metrics. Together ships with a pre-defined saved Audit set for the *Sun Code Conventions for Java,* which you can load and use in place of the default audit set, or any custom sets you create. For more information, see "Creating and using saved sets of metrics or audits" on page 287.

Note It is worth mentioning that both audits and metrics are only valid for source code that can compile. If your source code contains errors, or some libraries and paths are not included, audits and metrics might produce inaccurate results.

How to perform metrics analysis

Metrics analysis evaluates object model complexity to support quality assurance.

To perform metrics analysis:

1. Open a project.
2. Make sure that the Quality Assurance feature is activated (**Tools | Activate/ Deactivate Features** - Together *Features: Quality Assurance*).
3. Choose **Tools | Metrics** on the main menu.
4. Select the metrics you want to analyze. Each metric displays a description in the lower pane of the Metrics dialog box. For each metric, there are settings for options such as limits and granularity in the right-hand pane of the Metrics dialog box. Change the settings if necessary.
5. When you have selected your set of metrics, click **Start**.
6. The results are displayed on the Metrics tab in the Message pane. In the results table, select any element and choose **Open** from the right-click menu to navigate directly to it in the diagram or source code.

Metrics for audit violations

You can also run metrics on the most recent audit results. In the Metrics dialog, the available audits are listed in the *Audit Violations* group. The metrics for audit violations have the same abbreviations as the corresponding audits, only with the prefix *av*. The metrics for audit violations are calculated as the number of violations in each class for the selected audit. The total sum of audit violations can also be calculated (run the metric TAV).

How to perform audits

Audits automatically check for conformance to standard or user-defined style, maintenance, and robustness guidelines.

To perform audits:

1. Open a project.
2. Make sure that the Quality Assurance feature is activated (**Tools | Activate/ Deactivate Features** - Together *Features: Quality Assurance*).
3. Choose **Tools | Audits** on the main menu.

4. Select the audits you want to run. Each audit displays a description in the lower pane of the Audits dialog box. For each audit, the severity level (and other audit-specific options, in some cases) are displayed in the right-hand pane of the Audits dialog box. Change the settings if necessary.
5. When you have selected your set of audits, click **Start**.
6. The results are displayed on the Audits tab in the Message pane. In the results table, select any element and choose **Open** from the right-click menu to navigate directly to it in the diagram or source code.

Options for processing audits and metrics

The default processing settings are adequate for running audits and metrics. However, if you want to change the processing behavior, use the Options dialog (Tools | Options | <level>) to set options. The options are available in the QA node on the default and project levels.

The following options specify how to process audits and metrics:

- *Process in background* - This option is unchecked by default, because it is safer to run audits and metrics in the command thread. If you want to edit something while you are waiting for audits or metrics to be processed, check this option. However, use caution - never edit a package while it is being processed for audits or metrics (this leads to false results or exceptions in Together).
- *Process classes* - When running audits or metrics for an entire project, all the classes in the model are processed by default. It is possible to also process the classes specified in Search/Classpath in Project Properties. Check the options for the classes that you want to process: *Process classes in model*, and/or *Process classes from Search/Classpath*.

Output for audits and metrics

Audits and metrics display reports of analysis results in the Message pane. Metrics results are displayed as a table: the rows show the classes, packages, or diagrams that were analyzed, and the columns show the corresponding values of selected metrics. Audit results display a list of audit violations and where they occurred.

The results of both audits and metrics analysis are tightly connected with source code. From any line of the results table, you can navigate to the appropriate location both in the diagram and in the source code. Select the row in the results table that is of interest to you, and double-click it, or choose **Open** on the right-click menu. The same functionality is available for metrics on the bar graph (see "Bar graph" on page 283).

Sorting results

When viewing the output of audits or metrics, you will probably want to compare and organize the items in the results table. There are several ways to sort results.

- Sorting by one column.
  - To sort all the items according to the values for a specific column, right-click anywhere in the column and choose **Sort Ascending** or **Sort Descending**.
  - Alternatively, click on the column heading. The arrow in the heading displays whether sorting is ascending or descending. Click again to change the direction.
- Complex sorting.
  - To prioritize several columns for sorting, right-click anywhere in the table and choose **Sort**. In the Sort dialog, you can choose up to three columns to use for ordering the items.
  - Alternatively, hold down the Shift key and click on a column heading to add it to the sorting list. The columns selected for sorting are marked with the sorting order (1, 2, 3, and so on). Shift + click again to change the sorting direction. Clicking on any column heading (without Shift) cancels complex sorting and sets sorting to that column only.
- Grouping audit results.
  - To group items according to the current column, right-click in the Audit table and choose **Group By**. This enables you to organize the results by changing the relationship of rows and columns.

Updating results

You can update or refresh the results table from the table's right-click menu.

- Click **Refresh** to recalculate the results that are currently displayed.
- Click **Restart** to open the Audits or Metrics dialog, where you can re-select the set that you want to run, and change options if necessary.
- For audits, click **Refresh in Results** to check only the classes that are displayed in the results table. This does not re-run classes that did not produce any violations. This is useful for comparison after making corrections, and it is considerably faster than running audits on all the classes.
- For audits, click **Turn Off Selected Rules** to remove the selected audit rules from the results table after refreshing the results. This excludes the selected audits from the current set of audits to run, so that when you perform Refresh or Refresh in Results, these rules are ignored.

Exporting results

To save the results of metrics or audits in a separate file:

1. On the right-click menu of the results table, point to **Export**, and choose the scope of the results to export (**Entire Table** or **Selected Rows**).
2. In the Export Results to File dialog box, specify the target file location and name.
3. In the *Output type* drop-down list, select the format for the exported file:

Separated by tabs - The results table is saved in a text file with tabs between the columns.

Aligned with spaces - The results table is saved in a text file with columns aligned by spaces.

Generate HTML file - The results table is saved in table format in an HTML file.

Create multifile HTML report (metrics only) - The results table is saved as a set of HTML files, with hyperlinks for navigating through the project tree model (see "Creating linked HTML reports for metrics" on page 279).

Save in loadable format (metrics only) - The results table is saved in a *.mtbl file that can be re-loaded in Together independently of the project (see "Saving and loading metrics results" on page 280).

4. Click **Ok** to save the results in the specified location.

Creating linked HTML reports for metrics

For metrics results, it is possible to generate a report in HTML format as a set of linked HTML files. The set of files representing the results of metrics analysis includes separate files for each package and class involved in the analysis, a file that covers results for all packages, and a file for all classes. All these files are hyperlinked to each other. This makes it possible to navigate through the project tree model.

To generate a linked HTML report for metrics:

1. On the right-click menu of the results table, point to **Export**, and choose the scope of the results to export (**Entire Table** or **Selected Rows**).
2. In *Output type*, select *Create multifile HTML report*.
3. In *Report directory*, enter the path to the folder where you want to save the report files, or click the chooser button to select a path in the file chooser.
4. The settings under *HTML Options* specify where to save the set of HTML files that contain individual descriptions of each metric. These are the same descriptions that display in the Metrics dialog, and they can be referenced from the HTML report. By default, these files are saved in a folder called `\descriptions` under the directory that you specified for output. If you need to specify a folder other than the default, uncheck *Standard location*. (Click **Help** in the dialog for more details on how to set these options.)
5. Check *Launch viewer* to automatically open the report in the browser.
6. Click **Ok**. While viewing the report, use the links to see all classes or all packages. Click on a column heading (metric abbreviation) to view the full name and description for that metric.

Saving and loading metrics results

After you run a metrics analysis, you can choose to save the results and later view the results table independently of the project. You can also use results files to share metrics results with other Together users. If you do not save results, you cannot view them again after you close the results table. Saved metrics results have the .mtbl extension.

To save a set of metrics results:

1. Right-click anywhere in the results table and choose **Export | Entire Table** from the right-click menu.
2. In the Export Results to File dialog, specify the path to the directory where you want to save the file and give the file a name.
3. In *Output type*, choose *Save in loadable format*.
4. In *Comment*, type a brief description of the results table that you are saving. This text is displayed as the title of the table when you open the file. If you leave this field empty, the filename is used as the table title instead.

The file is saved with the `.mtbl` extension.

To load a saved set of metrics results:

1. Choose **Tools | Load Metrics Results** on the main menu. It is not necessary to have a project open.
2. In the Load Metrics Results dialog, navigate to the *.mtbl file that contains the results you want to view, and click **Ok**. You can choose any existing *.mtbl file; it does not depend on the project.

The results table opens in the Message Pane on the Metrics tab.

Note The subtab for the table displays the name or text that was typed in the *Comment* field when the file was saved. If there is no Comment, the short filename is used instead. When you focus on the name, a tool tip displays the full path and filename of the saved *.mtbl file.

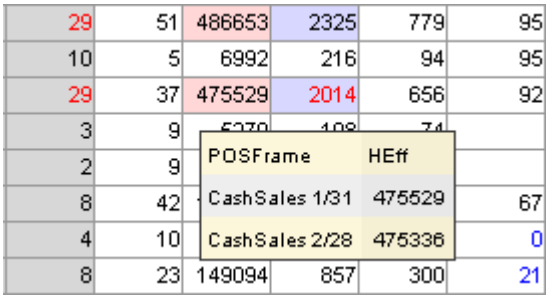
Comparing metrics results

If you have several sets of metrics results open, you can compare the values in one table against the values in another table. Use this feature for comparing projects or for comparing changes in a project over time.
Differences between results can be both highlighted in the table, and listed in tool tips, as shown in Figure 62.

- For comparing one table in relation to another, colors are used to highlight table cells with values that are higher or lower than the values in the other table.
- For comparing specific values in two or more tables, a tool tip over each table cell shows a list of tables with their values for the corresponding cell (that is, the results for the same metric in the same object).

You can also quickly navigate from a selected object in the current table to the same object in a different table, using the Go to Object in Table command.

Figure 62 Comparing metrics results



To compare tables:

1. Make sure that all of the tables you want to compare are open. The differences will be displayed in the table that is currently focused.
2. Right-click anywhere in the table and choose **Compare** on the right-click menu.
3. In the Compare dialog, you can optionally change the color scheme for highlighting differences. Use the color chooser buttons to make changes.
   - *Higher* is the color used to highlight table cells that have higher values than in the other table.
   - *Lower* is the color used to highlight table cells that have lower values than in the other table.
   - *Not found* is the color used to highlight table cells that do not exist in the other table.
   - If the values are the same, they are not highlighted.

4 In *Compare with*, the drop-down list displays all the open tables. Choose the table to visually compare results against. The box above displays an informational message if the second table does not have some of the columns that are in the first table.

   If you select *No comparison*, nothing will be highlighted.

5. Under *Show in Tool Tips*, select the names of the tables that you want to include in the tool tips for comparing cell values. Add them to the *Selected tables* box.

   If you do not want to display tool tips, leave the *Selected tables* box empty.

6. Click **Ok**.

To remove comparison highlighting and tool tips:

1 Right-click anywhere in the table and choose **Compare** on the right-click menu.

2. In *Compare with*, choose *No comparison* and click **Ok.**

The table is restored to the way it looked before you performed the comparison.

To find a corresponding object in another table:

1 Right-click anywhere in the row of the package, class, or operation that you want to find. Choose **Go to Object in Table** on the right-click menu.

2. The submenu displays a list of open tables. Choose the one you want to find the object in.

If the object exists in the results table you selected, the table opens, with the corresponding object selected. If the table does not open, it means that the object was not found.

Graphic output of metrics results

Metrics results can be viewed as graphic output. Right-click on a table cell that contains results for a metric, and choose a type of graph to display (Bar Graph, Distribution Graph, or Kiviat Graph).

Note Bar graphs and distribution graphs display only the nodes that are expanded in the results table. If it seems that some objects are missing in a bar graph or distribution graph, it is probably because they are hidden in the table. Use the +/- buttons to the left of the table to expand or collapse branches before generating a graph.

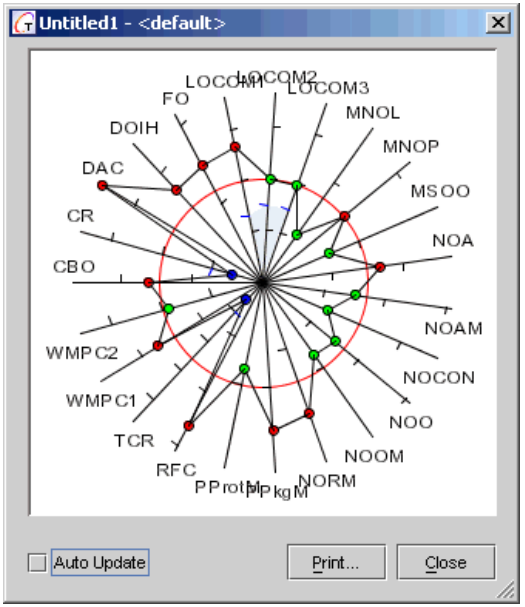
Kiviat graph

The Kiviat graph (see Figure 63) demonstrates the analysis results of the currently selected class or package for all the metrics that have pre-defined limiting values. The metrics results are arranged along the axes that originate from the center of the graph.

Each axis has a logarithmic scale with the logarithmic base being the axis metric upper limit, so that all upper limit values are equidistant from the center. In this way, limits and values are displayed using the following notation:

- Upper limits are represented by a red circle, showing that any points outside the red circle violate the upper limit.
- Lower limits are represented by blue shading, showing that any points inside the blue area violate the lower limit. Note that blue shading does not show up in areas of the graph with lower limits of 1 or 0.
- The actual metrics show up in the form of a star, with metric values drawn as points.
  - Green points represent acceptable values.
  - Blue points represent values below the lower limit.
  - Red points represent values exceeding the upper limit.
- Scale marks are displayed as clockwise directional ticks perpendicular to the Kiviat ray.
- Lower limit labels are displayed as counter-clockwise directional blue ticks perpendicular to the Kiviat ray.

Figure 63 Kiviat Graph



Bar graph

The bar graph (see Figure 64) displays the results of a selected metric for all packages, classes, and/or operations.

The bar color reflects conformance to the limiting values of the metric in reference:

- Green represents values that fall within the permissible range.
- Red represents values that exceed the upper limit.
- Blue represents values that are lower than the minimal permissible value.
- A thin vertical red line represents the upper limit, and a thin vertical blue line represents the lower limit.

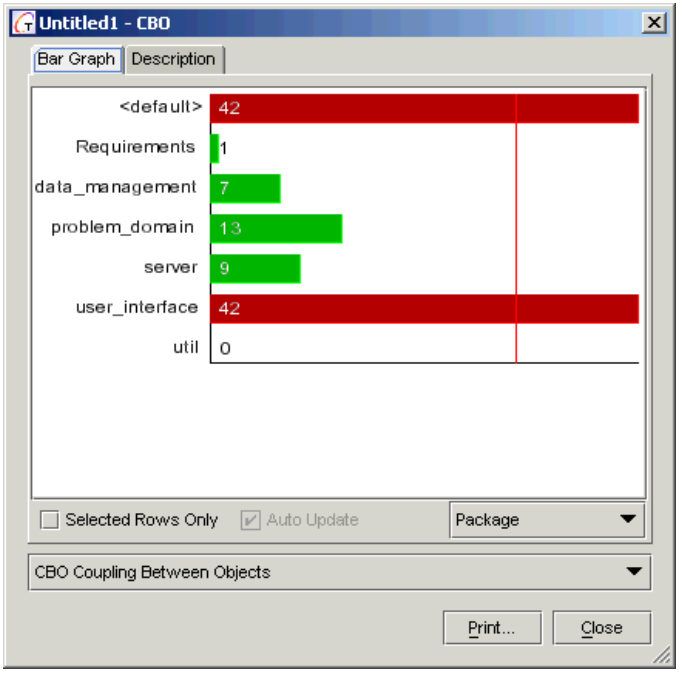
To change the metric to display results for, select from the drop-down list of metrics in the dialog box. You can refer to the explanation of the current metric on the Description tab.

To set the scope of the results to display, choose *Package*, *Class*, or *Operation* from the drop-down list in the dialog box.

To view results only for the highlighted rows of the table (instead of all rows), check *Selected rows only*. To browse through the table and view the graph, check *Auto update*. If *Auto update* is unchecked, the graph displays the original selected rows and does not change if you select different rows in the table.

You can navigate directly from the bar graph to the source code in the JBuilder Editor and the corresponding element in the Designer pane. Double-click on a bar in the graph, or right-click and choose **Open**. Switch focus to JBuilder to view the source code.

Figure 64 Bar Graph



Distribution graph

The distribution graph (see Figure 65) displays the distribution of values of all packages, classes, and/or operations for a selected metric. This graph is particularly useful for getting an overview of an entire project based on a specific metric.

A thin vertical red line represents the upper limit, and a thin vertical blue line represents the lower limit.

Check *Log X* to display the x-axis on a logarithmic scale. In most cases, this makes the graph easier to interpret correctly.

To change the metric to display results for, select from the drop-down list of metrics in the dialog box.

To set the scope of the results to display, choose *Package*, *Class*, or *Operation* from the drop-down list in the dialog box.

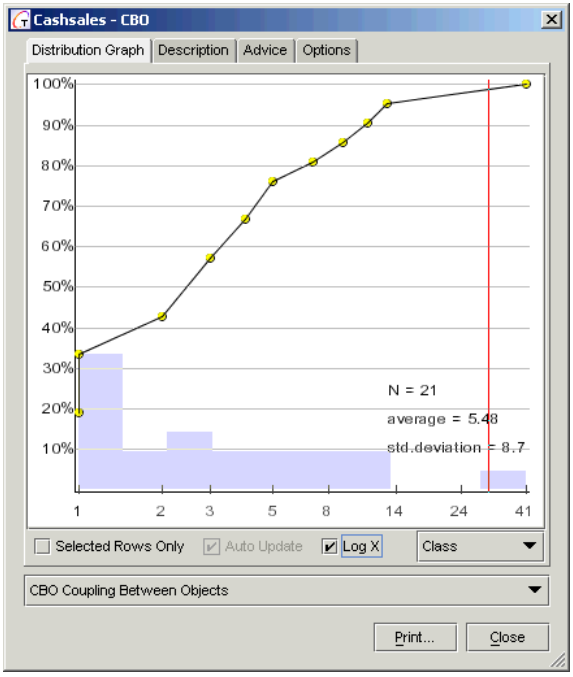
To view results only for the highlighted rows of the table (instead of all rows), check *Selected rows only*. To browse through the table and view the graph, check *Auto update*. If *Auto update* is unchecked, the graph displays the original selected rows and does not change if you select different rows in the table.

For a general explanation of the current metric, refer to the Description tab.

For tips on how to interpret the results and how to improve them, refer to the Advice tab. This tab also displays the extreme highs and/or lows for the current metric, in tables called *Top of metric* and *Bottom of metric*. These tables point you to packages or classes that you should consider revising.

To review the upper and lower limits and other options that are set for the current metric, refer to the Options tab.

Figure 65 Distribution Graph



Printing results

Print results either by printing the table displayed in the Message pane or by printing graphs when you view them.

Printing results from the table

To print the entire table, right-click anywhere in the table and choose **Print** from the right-click menu. In the Print dialog, you can preview the output and set the printing scale.

To print the results for specific metrics:

1. Right-click anywhere in the table and choose **Print** from the right-click menu.
2. Under *Select Columns to Print*, check the boxes corresponding to the metrics you want to print results for (all columns are checked by default, so you need to uncheck those metrics that you do not want to print).
3. Under *Scope*, check *Expand nodes* if you want to print the contents of collapsed nodes. If *Expand nodes* is not checked, only the items that are currently visible in the table will be printed.
4. Preview the output and set the printing scale if necessary.

To print the results for specific rows:

1 First select the rows in the table that you want to print.
2. Choose **Print** from the right-click menu. The Print dialog opens.
3. Under *Scope*, choose *Selected rows*.
4. Under *Scope*, check *Expand nodes* if you want to print the contents of collapsed nodes. If *Expand nodes* is not checked, only the items that are currently visible in the selected rows will be printed.
5. Under *Select Columns to Print*, check the boxes corresponding to the columns you want to print for the selected rows (all columns are checked by default).
6. Preview the output and set the printing scale if necessary.

Printing graphic results

All graphs can be printed. To print a currently displayed graph, click the **Print** button in the graph dialog box. In the Print dialog, preview the output and set the printing scale.

Automatic correction of audit violations

Some of the audit rules provide automatic correction for violations. This helps you quickly fix certain problems when you run audits to check your own code. In the results table, violations that can be automatically corrected are marked with the letter F in the *Fix* column. A lock symbol is displayed when a violation cannot be fixed because it is in a read-only file.

Table 35 lists the audits that currently provide automatic correction features.

To automatically correct violations:

1. In the table, select the rows with violations you want to fix.
2. Choose **Auto Correct** on the right-click menu.
3. For each violation, you are prompted to confirm automatic correction. In the Auto Correct dialog, click Yes to fix the violation that is displayed, or No to skip it.

4. Optionally set the scope of your confirmation under *Assume this answer for*. In this way, you can choose to either handle each violation individually or immediately fix or skip a group of violations.

The violations that have been fixed are marked with a green checkmark in the *Fix* column.

Table 35 Audits that have automatic correction features

Audit Group	Abbreviation	Audit Name
Declaration Style	CPASBF	Constant Private Attributes Should Be Final
Documentation	DCBA	Detect Collection-Based Associations
Possible Errors	CSF	Call super.finalize() from finalize()
Possible Errors	UL	Use 'L' Instead of 'l' at the End of Integer Constants
Superfluous Content	ILC	Import List Construction
Superfluous Content	OIM	Obsolete Interface Modifiers
Superfluous Content	UIMM	Unnecessary Interface Member Modifiers
Superfluous Content	UPCM	Unused Private Class Members

Creating and using saved sets of metrics or audits

The Metrics dialog and the Audits dialog display the set of all available metrics and audits respectively. When you open a project, a default subset of all available metrics and audits is active. Active metrics/audits are indicated by a checkmark. If you open one of the dialogs and click *Start*, all of the active metrics/audits are processed.

You will not want to run every metric/audit in the default active set every time, but rather some specific subset of available metrics/audits. Together allows you to create saved sets of active metrics and audits that can be loaded and processed as you choose. You can always restore the default active set using the *Set Defaults* button in the dialogs.

Use the default active Metrics/Audits set or any saved set as the basis for creating a new saved set. For example, you could load the saved audit set *SunCodeConventionsforJava.adt* and create a new saved Audits set based on it.

The default location for saved Metrics/Audits sets is: `$TGH$/modules/com/togethersoft/modules/qa/config`.

To create a saved set of active metrics/audits:

1. Open the relevant dialog (Metrics or Audits) as described in "Running audits and metrics in Together" on page 275.

2. If you want to base your new saved set on the default set, click **Set Defaults**. If you want to base it on a previously created custom set, click **Load Set**, then navigate to and select the desired saved set file (.adt for audit sets, .mts for metrics).
3. Go through the individual metrics/audits and check those you want to include in the set, or uncheck those you do not want to include. Select all the items in a group by checking the group name.
4. When you complete your selection, click **Save Set As** and specify the location and filename for the saved set file.

To use a saved set of active metrics/audits:

1 Open the relevant dialog (Metrics or Audits) as described in "Running audits and metrics in Together" on page 275.

2. Click **Load Set** and navigate to the saved set file you want to use.
3. Click **Start**.

Tip You might want to include the .adt and .mts files in your backup routine.

Audits and metrics supported for Java

Together provides over 100 audits and approximately 60 metrics for Java projects. Table 36 lists the metrics supported for Java and Table 37 lists Java audits.

Table 36 Java Metrics

AC	Attribute Complexity
AHF	Attribute Hiding Factor
AIF	Attribute Inheritance Factor
CBO	Coupling Between Objects
CC	Cyclomatic Complexity
CDBC	Change Dependency Between Classes
CF	Coupling Factor
CR	Comment Ratio
DAC	Data Abstraction Coupling
DOIH	Depth Of Inheritance Hierarchy
FO	FanOut
HDiff	Halstead Difficulty
HEff	Halstead Effort
HPLen	Halstead Program Length
HPVoc	Halstead Program Vocabulary
HPVol	Halstead Program Volume

Table 36 Java Metrics (continued)

LOC	Lines Of Code
LOCOM1	Lack of Cohesion of Methods 1
LOCOM2	Lack of Cohesion of Methods 2
LOCOM3	Lack of Cohesion of Methods 3
MHF	Method Hiding Factor
MIC	Method Invocation Coupling
MIF	Method Inheritance Factor
MNOL	Maximum Number of Levels
MNOP	Maximum Number Of Parameters
MSOO	Maximum Size of Operation
NOA	Number Of Attributes
NOAM	Number Of Added Methods
NOC	Number Of Classes
NOCC	Number Of Child Classes
NOCON	Number of Constructors
NOIS	Number of Import Statements
NOM	Number Of Members
NOO	Number Of Operations
NOOM	Number Of Overridden Methods
NOprnd	Number of Operands
NOprtr	Number of Operators
NORM	Number of Remote Methods
NUOprnd	Number of Unique Operands
NUOprtr	Number of Unique Operators
PF	Polymorphism Factor
PPkgM	Percentage of Package Members
PPrivM	Percentage of Private Members
PProtM	Percentage of Protected Members
PPubM	Percentage of Public Members
RFC	Response for Class
TAV	Total Audit Violations
TCR	True Comment Ratio
TRAu	Total Reuse from Ancestors unitary
TRAp	Total Reuse from Ancestors percentage
TRDu	Total Reuse in Descendants unitary
TRDp	Total Reuse in Descendants percentage
VOD	Violations of Demeters Law

Table 36 Java Metrics (continued)

WMPC1	Weighted Methods Per Class 1
WMPC2	Weighted Methods Per Class 2

Table 37 Java Audits

ACE	Assignments in Conditional Expressions
AFLV	Assignments to 'for' Loop Variables
AFP	Assignments to Formal Parameters
ASMDCN	Accessing Static Members by the Descendant Class Name
ASMO	Accessing Static Members through Objects
ASWL	Appending to String Within a Loop
BLAD	Bad Location for Array Declarators
BTJC	Bad Tag in Javadoc Comments
CA	Complex Assignments
CEV	Constants with Equal Values
CIUCFL	Complex Initialization or Update Clauses in 'for' Loops
CFPT	Comparing Floating Point Types
CLE	Complex Loop Expressions
CNSMFN	Class Name Should Match File Name
CPASBF	Constant Private Attributes Should Be Final (autofix)
CQS	Command Query Separation
CSF	Call super.finalize() from finalize() (autofix)
CVSBF	Constant Variables Should Be Final
DVIL	Declaring Variables Inside Loops
DVSS	Declaring Variables in Separate Statements
EBWB	Enclosing Body Within a Block
ECB	Empty Catch Blocks
EIAV	Explicitly Initialize All Variables
EJB_CL	(EJB) Creating Class Loaders
EJB_CONSOLE	(EJB) Console Output
EJB_FDESCR	(EJB) Directly Reading or Writing File Descriptors
EJB_FILES	(EJB) Accessing Files and Directories
EJB_IO	(EJB) Using AWT, SWING, Other UI APIs
EJB_JDBC	(EJB) Using JDBC API from Session Beans
EJB_NATIVE	(EJB) Loading Native Libraries
EJB_REFL	(EJB) Using Reflection
EJB_SEC	(EJB) Obtaining Security Policy Information

Table 37 Java Audits (continued)

EJB_SECOBJ	(EJB) Using Security Configuration Objects
EJB_SFACT	(EJB) Setting Socket Factory
EJB_SOCKET	(EJB) Listening on a Socket
EJB_SUBST	(EJB) Using Subclass and Object Substitution Features
EJB_THREADS	(EJB) Managing Threads
EOBA	Equality Operations on Boolean Arguments
HIA	Hiding Inherited Attributes
HISM	Hiding Inherited Static Methods
HN	Hiding Names
ILC	Import List Construction (autofix)
ICSBF	Instantiated Classes Should Be Final (autofix)
IMCM	Inaccessible Matching Constructors or Methods
LF	Long Files
LL	Long Lines
LPPMF	List Public and Package Members First
MLOWP	Mixing Logical Operators Without Parentheses
MSC	Multiple String Concatenations
MSOL	Multiple Statements on One Line
MVDWSN	Multiple Visible Declarations With Same Name
NC	Naming Conventions
NEC	Names of Exception Classes
NFSA	Non-Final Static Attributes
NLC	Numerical Literals in Code
NOIS	Negation Operator in ‘if’ Statement
ODCM	Order of Declaration of Class Members
OIM	Obsolete Interface Modifiers (autofix)
OM	Order of Modifiers
ONAMAM	Overriding a Non-Abstract Method with an Abstract Method
OPM	Overriding a Private Method
OSNBU	Operator ‘?:’ Should Not Be Used
OWS	Overloading Within a Subclass
PCO	Parenthesize Conditional in Operator ‘?:’
PDBB	Place Declarations at Beginning of Blocks
PFC	Provide File Comments
PIFS	Provide Incremental in ‘for’ Statement or Use ‘while’ Statement
PJDC	Provide Javadoc Comments
PMFL	Place Main Function Last

Table 37 Java Audits (continued)

POSNT	Place Operations with Same Name Together
PPA	Public and Package Attributes
PPCF	Place Public Class First
RIP	Referencing Implementation Packages
SAUI	Static Attribute Used for Initialization
SBCCS	Supply Break or Comment in Case Statement
SEB	Statements with Empty Body
SL	String Literals
SM	'synchronized' Modifier
SSSIDC	'switch' Statement Should Include a Default Case
DCBA	Detect Collection-Based Associations
TMSSC	Too Many Switch Statement Cases
UAAO	Use Abbreviated Assignment Operator
UC	Unnecessary Casts
UCVN	Use Conventional Variable Names
UE	Use 'equals' Instead of '=='
UIMM	Unnecessary Interface Member Modifiers (autofix)
UIOE	Unnecessary 'instanceof' Evaluations
UL	Use 'L' Instead of 'l' at the End of Integer Constants (autofix)
ULVFP	Unused Local Variables and Formal Parameters
UPCM	Unused Private Class Members (autofix)
URSP	Unnecessary Return Statement Parentheses
UTE	Use 'this' Explicitly to Access Class Members

Running audits and metrics from the command line

It is possible to run audits or metrics from the command line. This is useful for including quality assurance analysis as part of an automated daily build or other process. This section documents the command-line syntax and options.

Usage

The following syntax runs QA features in console mode (that is, without the GUI).

`TgStarter -script:com.togethersoft.modules.qa.QA [`*`options`*`] PrjName`
Where:

TgStarter is one of the following:

```
%TGH%/bin/Together.sh
%TGH%\bin\Together.bat
```

```
%TGH%\bin\TogetherCon.exe
%TGH%\bin\Together.exe -con
```
PrjName is a fully qualified project name, such as:

```
%TGH%\samples\java\CashSales\CashSales.tpr
```

Note Specify paths above using either a slash or backslash, as required by your operating system.

Options

Table 38 lists the options to use for running audits and metrics from the command line.

Table 38 Command Line Options

Name	Description
-?,-h,-help	Print this usage message and exit
-audit [out:*file*] [sort:[-]*column*] [cfg:*optset*]	Run audit process with specified parameters *file* = output file (default `%TgHome%/out/audit`) *column* = [*column name*]. If the column name is prefixed by a dash (-), it specifies the reverse order. severity abbreviation (default) explanation element item file line *optset* = file containing the previously saved set of options (default is `current.adt`).
-metrics [out:*file*] [sort:[-]*column*] [cfg:*optset*]	Run metrics with the specified parameters *file* = output file (default is `%TgHome%/out/metrics`) *column* = [*column name*]. If the column name is prefixed by a dash (-), it specifies the reverse order. abbr (metric abbreviation) item (default) *optset* = file containing the previously saved set of options (default is `current.mts`).
-pkg [pkg1 [pkg2 [...]]]	Process specified packages only
-cls [cls1 [cls2 [...]]]	Process specified classes only

Table 38 Command Line Options (continued)

Name	Description
-fmt:[*tab* | *align* | *html* | *report* | *loadable*]	Output format: *tab* = separate columns by tabs (default) *align* = align columns with spaces *html* = generate HTML file *report* = create HTML report (for metrics only) *loadable* = create loadable metrics results file (*.mtbl)
-dcpy:[*directory*]	Target folder where the descriptions are copied. This option is only valid for HTML format.
-dref:[*html-ref*]	Reference to this folder in HTML file. This option is only valid for HTML format.
-cmt:[*comment*]	Comment for loadable metrics result file (*comment* is the text to display as the table title). This option is only valid for loadable format.

Sets of options have the default extensions `.adt` for audits and `.mts` for metrics. If the set file name contains no path, it is searched for in the current directory first and then in the directory specified in project options (Tools | Options | Project Level | QA: *Path to current adt/mts files*). If it is not found in either of these directories, it is searched for in the directory where settings are stored by default (`%TGH%/modules/com/togethersoft/modules/qa/config`).

Extending the QA module

To some extent, you can customize individual audits and metrics by setting the options in the right pane of the Audits or Metrics window.

However, if you have more specific needs, you can create your own custom audit and metric plug-ins to extend the QA module.

The QA module resides in `$TGH$/modules/com/togethersoft/modules/qa`. This directory also includes detailed API documentation, samples for writing custom QA plug-ins, and source code examples.

To write your own custom QA plug-in:

1. Write Java source code for your custom audit or metric. For instructions and tips, refer to the API documentation for the QA module (`$TGH$/modules/com/togethersoft/modules/qa/doc/index.html`).

Tip Remember to include the following required libraries in the project classpath:

```
$TGH$/lib/openapi.jar

$TGH$/modules/com/togethersoft/modules/qa/qa.jar
```

2. Compile the Java class and create a new directory for it under the appropriate QA module subdirectory (under `/qa/audit/` for audit plug-ins, or under `/qa/metrics/` for metric plug-ins).

   The class name must be exactly the same as the name of the directory that you create for it. The package name for the plug-in must be equal to `com.togethersoft.modules.qa.<subsystem>.<plugin_name>`, where `<subsystem>` is either `audit` or `metrics` and `<plugin_name>` is the class name.

3. Write a description of your custom plug-in in an HTML file and place it in the directory that you created for the plug-in.

   The name of the description file must be the same as the class file, but with the .html extension.

4. If your custom plug-in is a metric, write a results tip in an HTML file and place it in the directory that you created for the plug-in. The tip should give advice on how to interpret the results of your metric. This text is displayed on the Advice tab on the Distribution graph.

   The name of the tip file must be equal to `<plugin_name>_Tip.html`, where `<plugin_name>` is the class name.

5. Launch a separate instance of Together for testing the plug-in.

Note To view messages for debugging the plug-in, you need to watch the Together console. Do not launch the `Together.exe` file, because it does not display the console. Use `TogetherCon.exe`, `Together.sh`, or `Together.bat` instead.

6. Run the corresponding QA feature (Audits or Metrics). Your custom plug-in should be displayed in the list of available audits or metrics.

Additional information sources

Shyam R. Chidamber and Chris F. Kemerer, *A metrics suite for object oriented design*. IEEE Transactions on Software Engineering, 20(6), pp476-493, 1994.

Thomas J. McCabe, *Complexity Measure*. IEEE Transactions on Software Engineering, Volume 2, No 4, pp 308-320, December 1976.

Arthur H. Watson and Thomas J. McCabe, *Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric*. Computer Systems Laboratory, National Institute of Standards and Technology, Gaithersburg, MD 20899-0001, September 1996.

Halstead, *Elements of Software Science*. New York, Elsevier North-Holland, 1977.

Brian Henderson-Sellers, *Object-Oriented Metrics: Measures of Complexity*. Prentice Hall, December 1995.

Object-Oriented Metrics: an Annotated Bibliography: **http://dec.bournemouth.ac.uk/ESERG/alpha.html**

Part

6

J2EE DEVELOPMENT

- Chapter 19, “J2EE Support”
- Chapter 20, “Creating EJB Components”
- Chapter 21, “Assembling EJB Modules”
- Chapter 22, “Container Transactions and EJB References”
- Chapter 23, “Designing and Developing Web Applications”
- Chapter 24, “Designing and Developing Enterprise Applications”
- Chapter 25, “Designing and Developing Application Clients”
- Chapter 26, “Designing and Developing Resource Adapters”
- Chapter 27, “J2EE Deployment”
- Chapter 28, “J2EE Platform Security Support”

Chapter

19

J2EE Support

Together provides deep support for the Java™ 2 Platform, Enterprise Edition (J2EE). With Together, you can create servlets, JSPs, EJBs, and Web or enterprise applications, and then deploy them to your current application server. This chapter gives an overview of the features that enable developing such multitier distributed applications.

All chapters in this part of the *User Guide* assume that you are familiar with fundamental J2EE concepts. (See *http://java.sun.com/j2ee* for extensive information on J2EE specifications and usage.) The topics covered in this chapter are:

- "Supported J2EE technologies" on page 299
- "Special J2EE diagrams" on page 300
- "References support" on page 304
- "Security support" on page 304

Supported J2EE technologies

Multitier enterprise applications are typically comprised of J2EE components such as servlets and EJBs that link to databases or implement business logic and HTML pages, JSPs, and applets that interface with the user. Developing an application requires creating its component pieces, writing deployment descriptor `*.xml` files, and then deploying an archive of the resulting files to an application server.

Together supports creating and deploying EJB modules, Web applications, and enterprise applications by:

1. Providing special J2EE diagrams. The diagrams enable you to model applications, organize the contents of JARs, WARs, EARs, and RARs, use tag libraries, set EJB references, and specify security mechanisms.
2. Creating deployment descriptors and archives based on diagrams and your current application server.
3. Deploying from diagrams.

Together automates deployment of the following major components:

- Web files (such as HTML and GIF)
- Servlets and JSP files
- JSP tag libraries
- Web applications
- EJBs and EJB modules
- Classes required for servlets, JSPs and EJBs
- Application clients
- Resource adapters

Together's e-commerce feature provides its J2EE support. This feature is activated by default. If the e-commerce feature is not activated for your configuration, choose **Tools | Activate/Deactivate Features** from the main menu and check **E-Commerce**.

Special J2EE diagrams

Together has several types of diagrams to support the J2EE platform. These are special to Together and are not among the canonical UML diagrams.

Together's J2EE diagram types are:

- EJB assembler diagram: for JAR files of J2EE components.
- Web application diagram: for WAR files of J2EE and Web components.
- Enterprise application diagram: for EAR files.
- Taglib diagram: for JSP tag libraries.
- Application client diagram: for applications that run on client machines.
- Resource adapter diagram: for RAR files.

You can deploy enterprise applications from class diagrams, Web application diagrams, EJB assembler diagrams, and enterprise application diagrams.

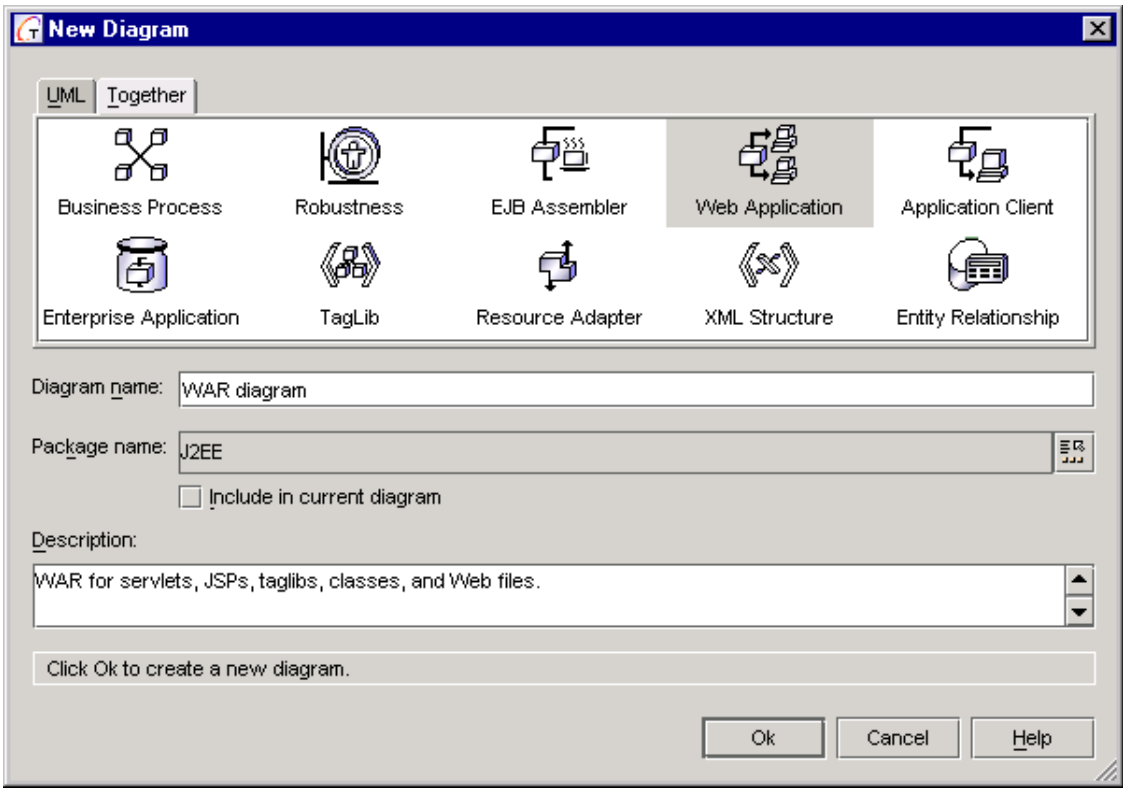
Creating a J2EE diagram

Creating a new J2EE diagram consists of four steps:

1. Click the **New Diagram** button ( ) on the Designer pane's horizontal toolbar.
2. Click the **Together** tab to reveal the non-UML diagram types.
3. Select the desired J2EE diagram type from the icon display. Figure 66 shows the dialog box for creating a new Web application diagram.
4. Fill in the diagram name and documentation as desired. Then click **Ok** to complete the process.

As an alternative to the steps listed above, you can use the Object Gallery to access "experts," which generate J2EE diagrams based on your input. The Object Gallery is available from the **File | New** command on Together's main menu. Select **Enterprise** to access e-commerce experts; select **Web** to access Web experts.

Figure 66 Creating a new Web application diagram



EJB assembler diagram

An EJB assembler diagram provides a visual representation for a JAR archive. The JAR stores all EJBs (entity, session, and message-driven beans) as well as the other classes such as exceptions and utility classes required for the EJBs.

To start developing an EJB module, create an EJB assembler diagram. Alternatively, use the Object Gallery to create a new EJB module, which automatically generates an EJB assembler diagram. For complete instructions on how to create use an EJB assembler diagram for developing EJB modules, refer to Chapter 21, "Assembling EJB Modules" Once you have created an EJB module, use the J2EE Deployment Expert to generate deployment descriptors and deploy the EJB module to your current application server. The J2EE Deployment Expert is available from the Deploy | J2EE Deployment Expert command on Together's main menu. See Chapter 20, "Creating EJB Components", and Chapter 21, "Assembling EJB Modules".

Web application diagram

A Web application diagram provides a visual representation of a Web application archive (WAR) that stores all JSPs, servlets, taglibs, classes, and Web files. Together uses the information on the Web application diagram to create the deployment descriptors for your current application server.

For complete instructions on how to create and use a Web application diagram to develop Web applications, refer to Chapter 23, "Designing and Developing Web Applications." After creating the diagram, you can use the J2EE Deployment Expert to generate a deployment descriptor and deploy the Web application to your current application server.

Enterprise application diagram

An enterprise application diagram enables you to combine different JARs, WARs and RARs into an enterprise application archive (EAR). Enterprise application diagrams are also useful in modeling global security constraints via security roles in the EJB/WAR modules.

For complete instructions on how to create an enterprise application diagram and use it for developing enterprise applications, refer to "Designing and Developing Enterprise Applications" on page 379.You can add shortcuts to application clients on an enterprise application diagram. From the diagram, you can use the J2EE Deployment Expert to generate a deployment descriptor and deploy the application to your current application server.

Taglib diagram

JSP technology has support for reusable modules called *custom actions.* This support is accessible in versions 1.1 and higher. Custom actions require custom tags; a *tag library* is a collection of custom tags. A taglib diagram is intended for using and developing JSP tag libraries. Such a diagram is convenient for tasks like message managing or accessing databases.

To develop a tag library or to use an existing one, start by creating a taglib diagram. For complete instructions on how to create and use a taglib diagram, refer to "Working with tag libraries" on page 370.

You cannot deploy directly from a taglib diagram. However, you can place shortcuts to taglib diagrams on Web application diagrams for later deployment.

Application client diagram

Application clients are programs that execute on users' machines. They typically have graphical user interfaces, and they depend on a container to provide system services. Application client diagrams are intended for design and development of application client programs.

For complete instructions on how to create an application client diagram and use it for developing application clients, refer to Chapter 25, "Designing and Developing Application Clients" on page 387.

You cannot deploy directly from an application client diagram. Instead, you can place shortcuts of application client diagrams on EJB assembler or enterprise application diagrams for later deployment.

Resource adapter diagram

A resource adapter diagram describes the connector architecture that defines a standard Service Provider Interface (SPI) for integrating the transaction, security, and connection management facilities of an application server with those of a transactional resource manager. Resource adapter diagrams are useful for integrating J2EE applications with Enterprise Information Systems (EISs) that are not relational databases.

Resource adapter elements provide access to any resources anywhere. Elements on a single resource adapter diagram can be archived into an RAR file. For complete instructions on how to create and use resource adapter diagrams, see Chapter 26, "Designing and Developing Resource Adapters" on page 393.

You can use the J2EE Deployment Expert to generate a deployment descriptor and deploy a resource adapter to your current application server.

References support

Together supports the following types of references:

- EJB references (references to other EJBs)
- Security references (references to possible user groups) with different access rights
- Resource references (references to possible resources)
- Environment references (references to constants in the environment)

For EJB 2.0 specification Together supports two additional types of references:

- EJB local references (references to other EJBs located in the same container)
- Resource-environment references

A reference is a special attribute in the EJB class. You can add references to an EJB by choosing **New | EJB Reference** on its right-click menu. Alternatively, you can add references and make changes on the Reference tab in the EJB's properties inspector.

The deployment descriptor, which is generated in the deployment process, saves information about references. An EJB can request the current status of transactions, security, links between EJBs, and so on, via the descriptor context.

For instructions on how to work with references, refer to Chapter 22, "Container Transactions and EJB References" on page 347.

Security support

J2EE security mechanisms specify security policies in an operational environment. Together supports J2EE security through special elements on J2EE diagrams. When you deploy from a J2EE diagram, Together's J2EE Deployment Expert places the information from these security elements into the deployment descriptor.

Most of Together's J2EE-oriented diagram contain security design elements.

- EJB assembler diagram: *Security Role*, *Method Permission*, *EJB Security Reference*
- Web application diagram: *Security Role*, *EJB Security Reference*
- EJB assembler diagram: *Security Role*
- Resource adapter diagram: *Security Permission*, *Authentication Mechanism*
- Application client diagram: *EJB Security Reference*

For instructions on how to work with security information, refer to Chapter 28, “J2EE Platform Security Support” on page 431.

Chapter

20

Creating EJB Components

This chapter explains how to design and develop enterprise Java beans. The chapter includes the following topics:

- "Setting the EJB properties for a project" on page 307
- "Creating EJB components" on page 308
- "Using EJB Inspectors to set EJB properties" on page 315
- "Working with EJBs" on page 327
- "Sharing home and remote interfaces" on page 333
- "Customizing the default EJB code" on page 335

Setting the EJB properties for a project

Every Together project is associated with an application server. The choice of application server determines the EJB properties available to a project. That choice does not determine which application server will be the ultimate target of deployment.

To choose an application server for an open Together project:

1. From the main menu, choose **Project | Project Properties**.
2. Click the **EJB** tab.
3. Choose the desired application server from the *Application Server* list.
4. Click **Ok** to quit the dialog and save the selection.

Note You can also set the EJB properties of a project through the options dialog. Choose **Tools | Options | <level>** from the main menu. Click the *EJB* node on the left, and pick the server in the *Application server* list on the right.

The EJB properties that the choice of application server impacts include:

- If the application server is EJB 2.0 compliant or Generic 2.0:
  - The class diagram toolbar contains a message-driven EJB button.
  - You can create local interfaces.
  - You can create application client diagrams and resource adaptor diagrams.
- The skeleton classes and interfaces that Together generates for EJBs and J2EE patterns vary according to whether the application server is EJB 1.0, 1.1, or 2.0 compliant.
- Some application server choices result in special EJB Inspector tabs.

To use encoding:

1. Choose **Tools | Options | <level>** from the main menu, and click the *EJB* node on the left.
2. Make sure that *Do not write encoding declaration for deployment descriptors* checkbox on the right is checked. Otherwise, you should know that encoding used for your project probably is not supported by the selected application server.

The J2EE Deployment Expert uses the application server for a project by default for the deployment target. You can choose a different target when you actually deploy. This makes it possible to deploy to two or more different application servers with the same project.

Creating EJB components

This section discusses the basics of viewing, editing, and creating EJB components.

Note Before creating EJBs, you should verify that the project's EJB properties are appropriate for your requirements. The code that Together generates depends on the choice of server and corresponding EJB specification.

Setting component editing and display options

The interfaces and classes of an EJB component act as a single entity, which is represented by the single implementation bean class.

By default, when you right-click an implementation bean class and choose **Delete**, **Cut**, **Copy**, or **Clone**, the interfaces and primary key classes are deleted, cut, copied, or cloned too. You can change that default behavior.

To change the behavior of Delete, Cut, Copy, Clone, or Paste of an EJB:

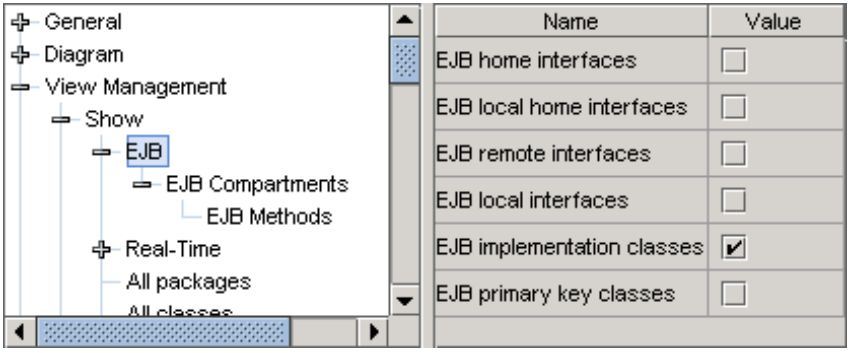
1. On the main menu, choose **Tools | Options | <level>** to open the Options dialog.
2. On the left pane, choose *EJB - Auto update related elements.*
3. On the right pane, check which component elements to clone, copy, paste, delete, or cut when applying the same action to the implementation bean.
4. Click **Ok** to save your choices.

By default, class diagrams display EJB implementation classes, but they hide interfaces and primary key classes. You can change the diagram options to display any of the EJB component parts.

To show or hide EJB component classes or interfaces:

1. On the main menu, choose **Tools | Options | <level>** to open the Options dialog.
2. On the left pane, expand *View Management - Show.*
3. Choose *EJB.*
4. On the right pane, check which component elements you want to display. Figure 67 shows the Options dialog box for displaying only the implementation classes.
5. Click **Ok** to save your choices.

Figure 67 Showing and hiding EJB component elements in the Options dialog



Although some EJB component part may be hidden in the Designer pane, the Explorer pane lists all parts. You can edit source code for any part by selecting it on the Explorer.

Setting component synchronization options

When you change the remote or home interface of an EJB, Together synchronizes the corresponding changes with the bean implementation class methods by default. For each method in the new interface that is not part of the original bean, Together creates a method stub in the bean class.

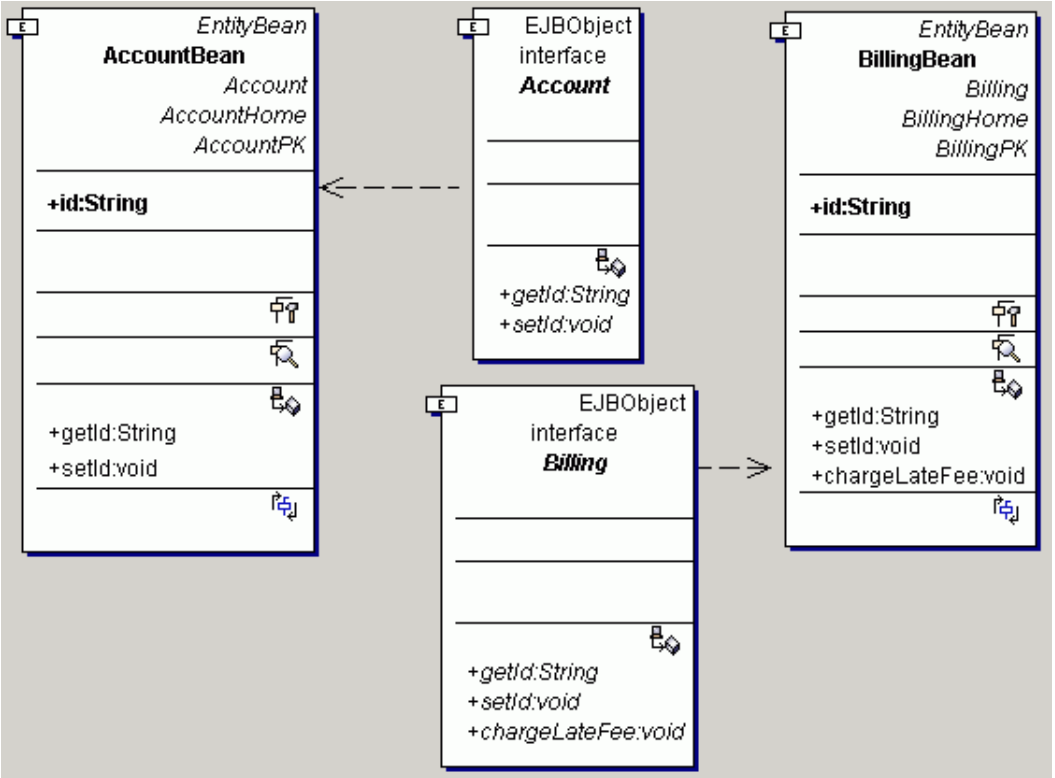
You can change that default behavior for automatic synchronization through the project or default options.

To change the EJB component synchronization properties:

1. On the main menu, choose **Tools | Options | <level>** to open the Options dialog.
2. On the left pane, choose *EJB - Synchronize methods when changing interfaces.*
3. On the right pane, un-check the component methods that you do not want to synchronize (bean class methods, remote or home interface or local interface methods).
4. Click **Ok** to save your choices.

The following example illustrates how Together synchronizes the changes. Figure 68 shows two entity EJBs, `AccountBean` and `BillingBean`. Notice the `chargeLateFee` business method for `BillingBean`.

Figure 68 Two entity beans with their corresponding remote interfaces



Continuing the example in Figure 68, suppose we change the remote interface for `AccountBean` to `Billing` by following these steps:

5. Right click `AccountBean` and choose **Properties** to open its Inspector.

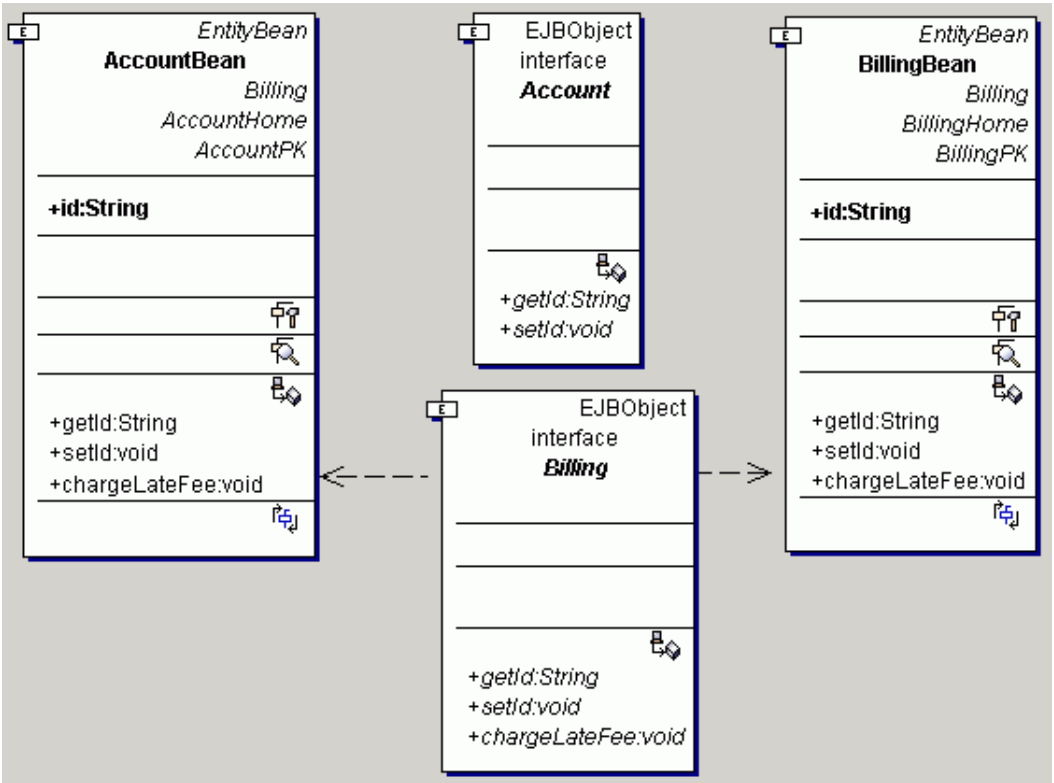
6. Choose the **Entity EJB** tab on the top, then choose the **General** tab on the bottom.
7. Click **Do not synchronize names**.
8. For the *Remote name* field**,** choose **Billing**.
9. In the resulting dialog box, click **Change class**.

Figure 69 shows the resulting class diagram with the new remote interface for `AccountBean`. Since synchronization with remote interfaces is turned on, Together provides this stub for the method `chargeLateFee` in the `AccountBean`:

```
public void chargeLateFee() throws RemoteException,
javax.ejb.EJBException {}
```

If the synchronization is turned off, Together does not create new methods such as `chargeLateFee` in the implementing bean to correspond to the methods in the new interface.

Figure 69 Entity EJBs sharing a remote interface



Creating EJB components from the class diagram

The class diagram toolbar has three different EJB buttons, corresponding to the three different kinds of EJB components. The toolbar buttons are listed in Table 39.

When you create an EJB component from the toolbar, Together provides skeletons for the home and remote interfaces, the bean, and the primary key class (for entity beans). The code is consistent with the application server specified in the project properties.

Table 39 EJB buttons on the class diagram toolbar

Button	EJB component type
	Entity EJB.
	Session EJB.
	Message-driven EJB. This button displays only when the project application server is EJB 2.0 compliant (or Generic 2.0).

Entity and session EJBs that satisfy the EJB 2.0 specification can have local interfaces. Together does not generate these interfaces when you create a new EJB.

To create local interfaces, right click the bean on the class diagram and choose New | Create Local Interfaces from the right-click menu. The two new local interfaces have the same method signatures as their non-local counterparts.

Creating EJB components using the Object Gallery

The Object Gallery provides EJBs among its collection of objects.

To create a new EJB using the Object Gallery:

1. On the main menu, choose **File | New**. This displays the Object Gallery.
2. Choose **Enterprise** among the categories listed on the left.
3. Choose the **EJB** icon on the right, and click **Next**.

   The resulting pages provide an expert to guide you through the process of creating the EJB, where you can specify the following values:

   - **EJB name.** The name of the EJB to be shown on the diagram.
   - **EJB display name.** The name of the EJB to use in the deployment descriptor.
   - **EJB type.** Entity, session, or message-driven.
   - **Package location.** The `<default>` package is the default location. You can click the file-chooser button to specify an existing package.

- **Options.** Check this to place a shortcut to the bean on an existing EJB assembler diagram. (A complete discussion on EJB modules is in Chapter 21, "Assembling EJB Modules.")

4. Click **Finish** if the bean is message-driven. Otherwise, click **Next** to set type-specific properties for entity or session beans.
   - **JNDI name.** Common to entity and session beans.
   - **Session bean.** Choose **Stateful** or **Stateless**.
   - **Entity bean.** Choose persistence type and check for simple primary key. If the bean has container-managed persistence, enter the **Pool name** and **Table name**.
5. Click **Finish** to close the dialog and generate the bean code.

Applying an EJB pattern to an existing class

You can change an existing class into an EJB implementation class by applying an EJB pattern.

To apply an EJB pattern to an existing class:

1. Right-click the class and choose **Choose Pattern**.
2. In the left pane of the resulting dialog box, expand *EJB*.
3. Choose the desired bean type: *Entity EJB*, *MessageDriven EJB*, or *Session EJB*.
4. Click **Finish** to close the dialog and generate the bean code.

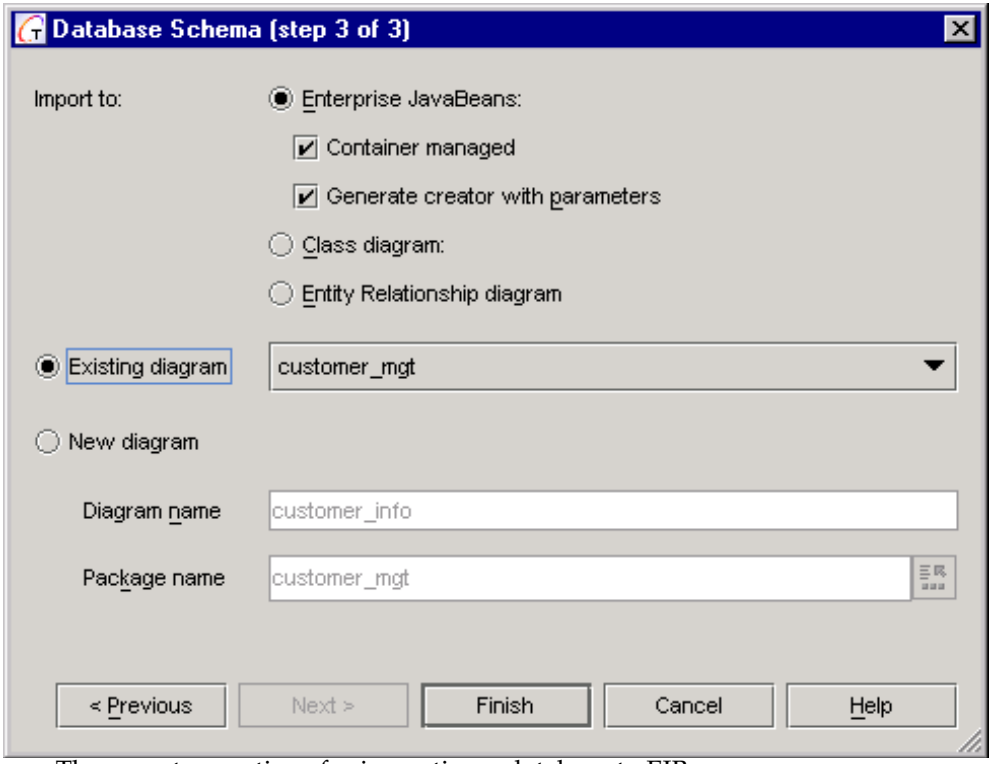
Applying the EJB pattern to a class appends `Bean` to the class name. Together creates remote and home interfaces and primary key classes for entity EJBs. Together also adds new members to the bean class appropriate for the project's targeted application server.

Applying an EJB pattern to an existing class may modify some of the original class members. For example, Together removes constructors and adds the appropriate `throws` statements to other operations.

Importing entity EJBs from a database

You can import a database to create new entity EJBs in an existing Together project. Chapter 6, "Importing and Exporting Information" has instructions for importing a database. Figure 70 shows the concluding dialog box for importing a database.

Figure 70 Importing entity EJBs from a database



There are two options for importing a database to EJBs:

- **Container managed.** Check to create a bean with container managed persistence. Leave it unchecked to create a bean with bean managed persistence.
- **Generate creator with parameters.** Check to generate an `ejbCreate()` method in which all columns of the database table are parameters. For example, if the source table has fields `CHAR column1` and `INT column2`, then the method signature is `ejbCreate(String, int)`.

When you import an EJB from a database, Together generates the code for an entity EJB component and places the component elements on the diagram. The diagram shows dependencies from the remote interface, the home interface, and the primary key class to the bean implementation class. The code that Together creates is consistent with the application server for the project.

Reverse engineering EJB source code

You can create a new project around existing EJB source code by reverse engineering. When you reverse engineer EJB source code, Together displays the EJB components on their package diagrams. (For instructions on how to re-engineer existing code, see Chapter 5, "Reverse Engineering.")

Using EJB Inspectors to set EJB properties

EJB inspectors have special EJB tabs. To open the Inspector, right-click the bean and choose **Properties** from the menu. When you click the **EJB** tab at the top, the special tabs display at the bottom. These tabs are for specifying general EJB properties, adding and removing fields and methods, and defining references, relationships, and deployment properties.

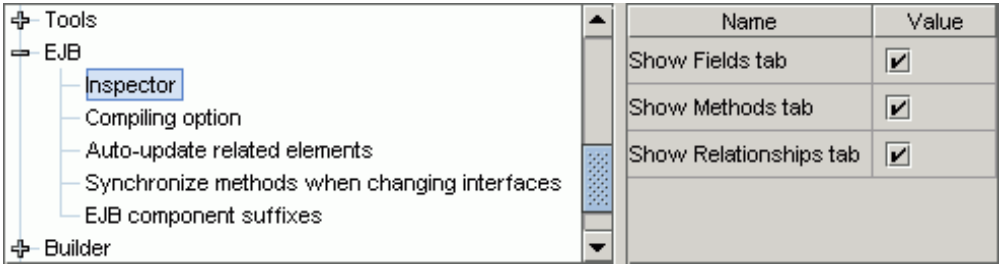
Configuring EJB inspectors

All EJB inspectors have **Deployment properties** and **EJB references** tabs. Session and entity EJB inspectors also have a **General** tab. You can hide or display three other EJB tabs: **Fields, Methods,** and **Relationships.**

To hide or display the optional tabs:

1. From the main menu, choose **Tools | Options | <level>.**
2. On the left pane, expand *EJB.* Then choose *Inspector,* as shown on Figure 71.
3. On the right, check or clear *Fields, Methods,* and *Relationships* to display or hide the corresponding subtabs in EJB inspectors.

Figure 71 Setting EJB Inspector tabs in the Project Options



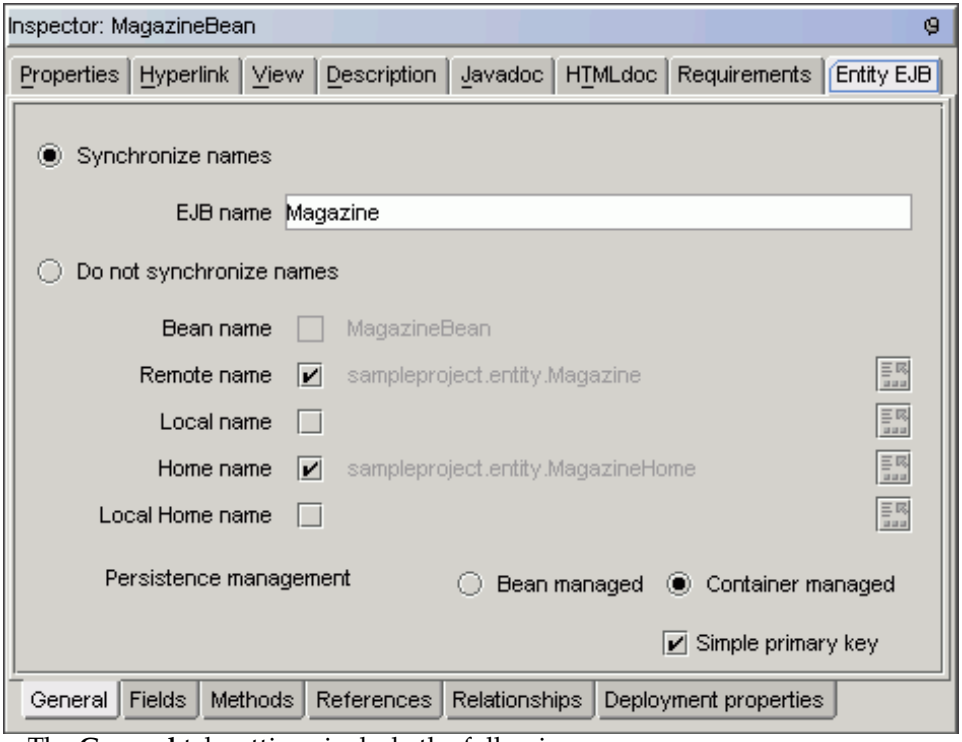
EJB Inspector tabs

The Inspector EJB tabs vary depending on the bean type, the current target application server for the project (EJB properties), and the Inspector configuration settings described above. This section discusses all tabs.

General tab

The **General** tab contains settings for naming component parts and for specifying persistence and primary key mapping. This tab is not present in inspectors for message-driven beans. Figure 72 shows the **General** tab for a container-managed entity bean named MagazineBean.

Figure 72 General tab of an entity EJB Inspector



The **General** tab settings include the following:

- **Synchronize names.** Default naming convention. If you change the EJB name, the names of the interfaces change automatically to match the new name.
- **Do not synchronize names.** Alternative naming convention for specifying interface names independent from the bean name. The text fields for entering the names are available only when you choose this setting.

  Remote interfaces are checked by default. *Local name* and *Local Home name* are enabled only when you are working with EJBs satisfying the EJB 2.0 specification.
- **Stateful.** (Session beans only) checkbox for specifying stateful or stateless session bean.
- **Persistence management.** (Entity beans only) default value is *Bean managed*.

- **Simple primary key.** A check indicates that there is a single field in the bean for the primary key. There are two ways to specify the primary key class for an entity bean with container-managed persistence:
  - Primary key maps to a single field of the entity bean class.
  - Primary key maps to multiple fields of the entity bean class (for implementation of compound keys via the user-defined `PKclass`).

  To create a primary key of the first type, delete the created `PKclass`. Then check *Simple primary key,* which is enabled for container-managed beans if there is only one primary key field with the appropriate object type. This helps avoid wrapping of the simple type primary key (such as `String`) into a user-defined class.

References tab

The **References** tab is for viewing existing references, changing their properties, removing them, and creating new references.

Radio buttons at the top of the **References** tab indicate which type of reference is currently displayed. Each display consists of a table with a row for each reference of the current type and columns for the properties.

To add a new reference or remove an existing one:

1. Click the radio button corresponding to the kind of reference.
2. To add a new reference, click **Add**. This creates a new line in the table where you can edit the reference properties.
3. To remove a reference, choose its row in the table and then click **Remove**.

There are six kinds of references:

- **EJB**: EJB references. Some of the EJB reference fields are common to other reference types.
  - **Name.** Name of the bean attribute that corresponds to the reference.
  - **EJB reference name.** Specific name to use for look-up.
  - **EJB reference.** Type of EJB that is referenced (session or entity). If you use the remote interface or home interface for the referenced bean, then the value of this column is filled in automatically.
  - **Remote interface.** Remote interface of the referenced EJB. This column has a chooser button for selecting the remote interface of the referenced bean.
  - **Home interface.** Home interface of the referenced EJB.
  - **EJB link.** Parameter in the deployment descriptor.
  - **EJB JNDI name.** Parameter in the deployment descriptor.

- **EJB local.** Local references. EJB local is visible only if the EJB satisfies the EJB 2.0 specification (that is, if you choose Generic 2.0, or an EJB 2.0 compliant application server for the project). The columns are the same as those listed above.
- **Resource.** Resource references. There are five columns for each resource reference. The columns *Name, Resource reference name,* and *Resource JNDI name* are analogous to their counterparts for EJB references. The two additional columns are:
  - **Resource reference type.** Data source type of the resource. This column has a list of data source types: `javax.sql.Datasource`, `javax.jms.QueueConnectionFactory`, `javax.jms.TopicConnectionFactory`, `javax.sql.XADatasource`, `javax.mail.Session`, and `javax.net.URL`.
  - **Resource reference authentication.** Specifies *Application* or *Container* authentication.
- **Resource-environment.** References to the container (local) resources. This type is visible only if the EJB satisfies the EJB 2.0 specification. *Name, Resource reference name,* and *Resource reference type* columns are analogous to their Resource reference counterparts described above.
- **Environment.** Environment references. The columns *Name* and *Environment entry name* are analogous to *Name* and *EJB Reference name* for EJB references. There are two additional columns:
  - **Environment entry type.** The type of an environment reference. This list consists of `java.lang` types: `Boolean`, `String`, `Integer`, `Double`, `Byte`, `Short`, `Long`, and `Float`.
  - **Environment entry value.** The value of the entry.
- **Security role.** Security role references. The column *Name* is analogous to the *Name* for ordinary EJB references. There are two additional columns:
  - **Role name.** The specific name to use for look-up.
  - **Role link.** An attribute in the deployment descriptor.

Deployment properties tab

The **Deployment properties** tab is for defining the major EJB properties required for the deployment process and used in the deployment descriptor. The properties on this tab vary according to the type of bean and the target application server for the project. See http://java.sun.com/dtd/ejb-jar_2_0.dtd for more details.

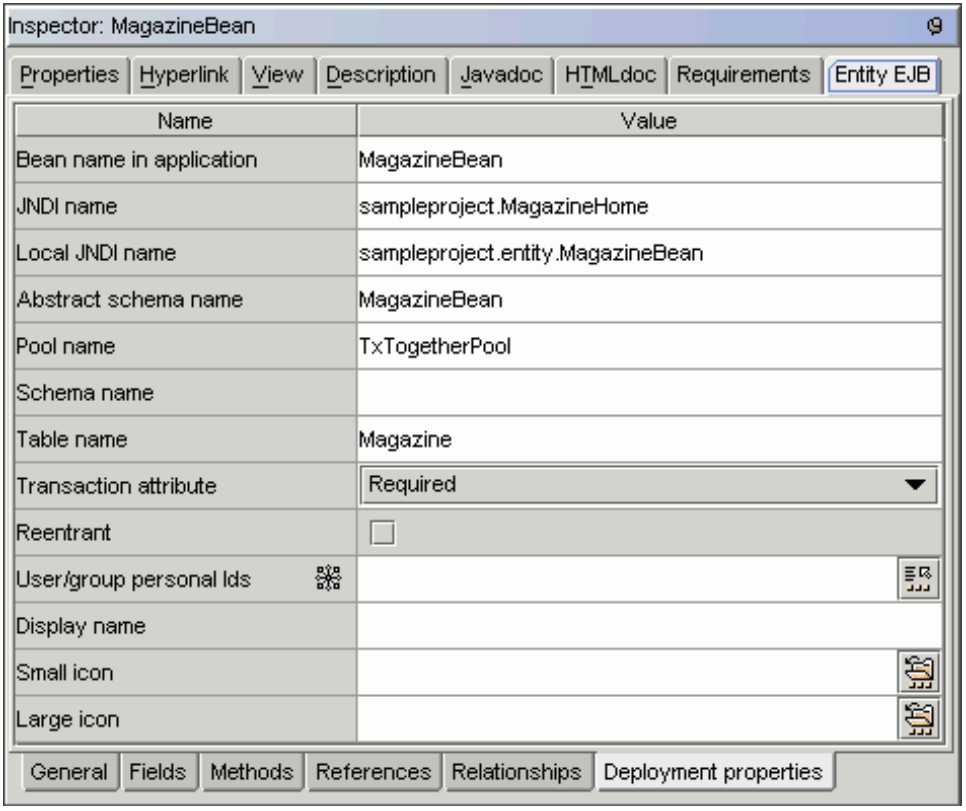
Figure 73 shows the Deployment properties tab for an entity EJB in the Magazine sample that ships with Together.

There are several categories of Deployment properties:

- External access fields specify how the bean is named.

- **Bean name in application.** The bean name in the deployment descriptor. This has the same default meaning as *Bean name* on the General tab.
- **JNDI name.** The JNDI name for a bean's home interface.
- **Local JNDI name.** The JNDI name for a bean's local home interface. This property is enabled only for entity and session beans and if the project satisfies the EJB 2.0 specifications. If a bean has both a remote home and a local home interface, then you must specify two JNDI names, one for each interface.

Figure 73 Deployment properties tab in the entity EJB Inspector



- Database settings (entity beans only) are four fields whose meanings vary according to the particular application server. All of the fields are disabled for BMP beans and enabled for CMP beans. They include:
  - **Abstract schema name.** Defaults to the bean name.
  - **Pool name.**
  - **Schema name**
  - **Table name.**

- Transaction settings consist of three fields:
  - **Transaction attribute.** Has the following possible settings: *NotSupported, Supports, Mandatory, Required, RequiresNew, Never.*
  - **Transaction management type.** (Session and message beans only) has two settings: *Bean* and *Container.*
  - **Reentrant.** (Entity beans only) should be checked if this entity EJB allows reentrant calls.
- Display and user settings are optional fields. They include:
  - **User/group personal IDs.** For inserting personal identifiers. This field has a multi-string editor.
  - **Display name.** A parameter for the deployment descriptor.
  - **Small icon.** For selecting the small icon (16x16 pixels `*.gif` or `*.jpg` image) for the application server's icon.
  - **Large icon.** For selecting the large icon (32x32 pixels `*.gif` or `*.jpg` image) for the application server's icon.

Fields tab

The **Fields** tab is available for entity beans only. This tab is for adding, editing, or deleting fields in the entity bean. The different fields display in a table, with rows for fields and columns for field properties.

To add a new field, click **Add** at the top of the tab. To remove a field, select the field and click **Remove**.

Note You can also add a new field to an entity bean directly from the diagram by right-clicking the bean and choosing **New | Field**. You can delete a field using **Delete** on the right-click menu of the field.

Each field has six properties:

- **Type.** The type of the field. You can edit the type from the list (`String`, `boolean`, `byte`, `char`, `double`, `float`, `int`, `long`, `short`, `java.util.Hashtable`, `java.util.Vector`) or pick the type from the corresponding chooser button.
- **Name.** The name of field member in the bean class.
- **Column.** (CMP beans only) the name of the field in the database.
- **Label.** The representation in a JSP client.
- **Primary key.** Check if this field is used as a primary key.
- **Relationship.** (CMP beans only) check if the field is used in a container-managed relationship. See "Creating container-managed relationships" on page 329.

Methods tab

The **Methods** tab is available for entity and session beans. It is for adding, modifying, or removing methods. You can add a new method on this tab by clicking **Add** at the top of the tab. You can remove a method by selecting the method and clicking **Remove**.

Note You can add a method to a bean directly from the diagram by right-clicking the bean and selecting **New | <type of method>**. You can delete a method using **Delete** on its right-click menu.

When you choose **New | Business method**, Together adds the business method to the remote interface as well as the bean class. If you do not want the method to be added to the remote interface, choose **New | Method** instead.

Radio buttons at the top of the Methods tab indicate which type of methods are currently displayed. There are five kinds of methods:

- **Creators.** Three creator properties are:
  - **Method signature.** Signature in the bean class.
  - **Exposed in Home.** Check if this method is accessible through the bean's home interface
  - **Exposed in Localhome.** Check if this method is accessible through the bean's local home interface. This property is enabled only if the target server is Generic 2.0 or EJB 2.0 compliant and if the bean has a local home interface.
- **Finders.** (Entity beans only) Properties are the same as creator method properties. There are two additional Finder properties:
  - **Multi.** Check if the finder can return many elements.
  - **Finder query.** The corresponding query.
- **Business methods.** Four business method properties are:
  - **Method signature.** Signature in the bean class.
  - **Exposed in remote interface.** Check if this method is exposed in the bean's remote interface.
  - **Exposed in local interface.** Check if this method is accessible through the bean's local interface. This property is enabled only if the target server is Generic 2.0 or EJB 2.0 compliant and the bean has a local interface.
  - **Transaction attribute.** A list consisting of *<bean default>, NotSupported, Supports, Mandatory, Required, RequiresNew, Never.*
- **Home methods.** (Entity beans only) Available only for entity EJBs satisfying the EJB 2.0 specification (the target server is Generic 2.0 or EJB 2.0 compliant). The home method properties are identical to the properties of creator methods.

- **Selectors.** (Entity beans only) Available only for CMP entity beans that satisfy the EJB 2.0 specification and that have simple primary keys. The Method signature and Finder query properties are the same as for finder methods.

Note According to the EJB 2.0 specification, all methods except selector methods can be exposed either in a remote interface or in a local interface.

Relationships tab

The **Relationships** tab is available only for CMP entity beans satisfying the EJB 2.0 specification. The relationships display in a table with rows for the relationships and columns for the relationship properties. There are seven properties:

- **CMR field name.** The name of the container-managed relationship as virtual field in the entity bean class.
- **Relationship name.** A unique name for a relationship.
- **Role name.** An optional name for the relationship role.
- **FK column.** A foreign key name in the database.
- **Key column.** A key field name in the database.
- **Multiplicity.** Multiplicity of the role, with possible settings *one* and *many.*
- **Relationship destination.** The path to the related EJB element.

For additional information, see "Creating container-managed relationships" on page 329.

Setting additional deployment properties

Some EJB inspectors have special tabs corresponding to the more popular application servers. The purpose of these additional tabs is to provide access to optional deployment properties for the EJB.

You can specify additional deployment properties for the following application servers:

- SunEE 1.3, Reference Implementation (*SunEE AS Properties* tab)
- Borland Enterprise Server 5.2 (*Borland Enterprise Server 5.2* tab)
- Bea WebLogic 6.0, 6.1, 7.0 (*WebLogic x.x* tab)

SunEE AS Properties Inspector tab

If the target application server for your project is SunEE 1.3, Reference Implementation, the EJB Inspector for CMP entity bean displays the **SunEE AS Properties** tab.

Check **Create a table on deploy** if you want to automatically create a table while deploying (`<table>` tag).

Check **Delete a table on undeploy** if you want to automatically remove a table while undeploying.

You can also write EJB QL queries in the following fields:

- **SQL statement used to create a table for an CMP EJB**
- **SQL statement used to delete a table for an CMP EJB**

The information contained in these fields is used while generating the EJB module deployment descriptor.

Borland Enterprise Server 5.2 Inspector tab

If the target application server for your project is Borland Enterprise Server 5.2, the inspectors for both EJBs and EJB modules display the **Borland Enterprise Server 5.2** tabs. The contents of these tab are dependent on the EJB type.

Note For additional information on the contents of the Borland Enterprise Server 5.2 tags, see **http://info.borland.com/techpubs/books/bes/pdfs52/**.

Properties is the common field for session, entity, message-driven beans, and EJB modules (EJB assembler diagrams). According to this information, Together generates corresponding custom `<property>`tags in the deployment descriptor.

Every property contains the following information:

- **Name** - a property name
- **Type** - a property type, for example, `java.lang.String`
- **Value** - a property value

Refer to the Borland Enterprise Server 5.2 documentation for a complete list of properties.

Borland Enterprise Server 5.2 Inspector tab for session EJBs

The Borland Enterprise Server 5.2 Inspector tab for a session EJB in addition contains the **Session timeout** (`<timeout>` tag) optional deployment property: time of saving the user session.

Borland Enterprise Server 5.2 Inspector tab for entity EJBs

The **Borland Enterprise Server 5.2** Inspector tab for CMP beans includes the following additional fields and checkboxes:

- **Create table** .`<table>`. Check if you want a table to create automatically while deploying.

- **Table properties**. Using this field, you can define different parameters of the database table. Refer to the Borland Enterprise Server 5.2 documentation for a list of parameters.

Borland Enterprise Server 5.2 Inspector tab for message-driven EJBs

The **Borland Enterprise Server 5.2** Inspector tab for a message-driven EJB in addition contains the **Connection Factory name** (a tag `<connection-factory-name>`) optional deployment property: the JNDI name of the JMS ConnectionFactory that the bean uses for its queues and topics.

WebLogic Inspector tabs

If the target application server for your project is BEA WebLogic 6.0, 6.1, or 7.0, some of the EJB inspectors display special **WebLogic** tabs. The contents of these tabs as well as whether the tabs are even present vary according to the EJB type and the version of the BEA WebLogic application server.

Note For additional information on the contents of the WebLogic Inspector tags, see http://e-docs.bea.com/wls/docs70/ejb/.

WebLogic 6.0 and WebLogic 6.1 Inspector tabs

If the target application server for your project is BEA WebLogic 6.0 or 6.1, entity EJB inspectors display a tab with the label **WebLogic 6.0** or **WebLogic 6.1.** This tab is present for entity EJBs only. There are no such tabs for session or message-driven EJBs.

There are two properties on the WebLogic 6.0/6.1 tab for an entity EJB Inspector:

- **Max beans in free pool.** <max-beans-in-free-pool> The maximum size of the free pool of beans that WebLogic maintains for this bean class.
- **Initial beans in free pool.** <initial-beans-in-free-pool> The initial size of the free pool of beans that WebLogic maintains for this bean class.

WebLogic 7.0 Inspector tabs

If the target application server for your project is BEA WebLogic 7.0, all EJB inspectors, display **WebLogic 7.0** tabs. This is true for session beans, entity beans, and message-driven beans

The view of this tab and the number of pages with the corresponding subtabs inside the WebLogic 7.0 tab vary among the different kinds of beans.

WebLogic 7.0 Inspector tab for session EJBs

The **WebLogic 7.0** Inspector tab for a stateless session EJB contains exactly the same settings as the WebLogic 6.0/6.1 Inspector tab for an entity EJB. For more information, see "WebLogic 6.0 and WebLogic 6.1 Inspector tabs" on page 324.

The **WebLogic 7.0** Inspector for stateful session beans gives access to the following optional deployment properties:

- **Maximum beans in cache.** <max-beans-in-cache> The maximum number of stateful session beans that are allowed in memory. Beans removed from the cache are passivated.
- **Idle timeout seconds.** <idle-timeout-seconds> The number of seconds that a stateful session bean can remain idle in an LRU cache. The default value is 30.
- **Cache type.** <cache-type> Choose the cache type from the list:
  - **NRU** (Not recently used): Stateful session beans are removed from the cache when the number of beans in the cache approaches <max-beans-in-cache>.
  - **LRU** (Least Recently used): Stateful session beans are removed from the cache after reaching <idle-timeout-seconds>.

WebLogic 7.0 Inspector tabs for entity EJBs

The **WebLogic 7.0** Inspector tab for entity EJBs is divided into two subtabs for a BMP bean: ***General*** and ***Cache.*** CMP bean inspectors have those two subtabs plus two more: ***Concurrency*** and ***Automatic key.*** The ***General*** and ***Cache*** subtabs for BMP beans are the same as for CMP beans. All subtabs display at the bottom of the Inspector window.

The **General** subtab is identical to the WebLogic 6.0/6.1 tab for entity EJBs. It has the fields *Max beans in free pool* and *Initial beans in free pool.* For more information, see "WebLogic 6.0 and WebLogic 6.1 Inspector tabs" on page 324.

The **Cache** subtab gives access to the optional deployment properties associated with cache memory:

- **Maximum bean in cache.** <max-beans-in-cache> Maximum number of entity beans of this type that are allowed in memory.
- **Idle timeout seconds.** <idle-timeout-seconds> Maximum amount of time an entity bean of this type is allowed in memory.
- **Read timeout seconds. <**read-timeout-seconds> The number of seconds between `ejbLoad` calls on a read-only entity bean. If this value is 0, the WebLogic Server calls `ejbLoad` only when it brings the bean into the cache.
- **Cache between transactions.** <cache-between-transactions> A check to cache the persistent data of an entity bean between transactions. The default is unchecked, which means caching occurs only during an individual transaction.

WebLogic 7.0 Inspector tabs for CMP entity EJBs

The WebLogic 7.0 Inspector tab for CMP beans includes the subtabs ***Concurrency*** and ***Automatic key*** in addition to the *General* and *Cache* subtabs.

The **Concurrency** subtab gives access to the optional deployment properties associated with the strategy for managing concurrent accesses to an entity bean. This subtab has three fields, *Concurrency strategy*, *Verify columns*, and *Optimistic column*.

- **Concurrency strategy.** <concurrency-strategy> Choose the strategy from the list:
  - **ReadOnly**: (For read-only entity beans) Creates a new instance for each transaction.
  - **Exclusive**: Only the EJB container can update the underlying data during a transaction.
  - **Database**: Allocates a separate entity bean instance for each transaction. The database handles locking and caching.
  - **Optimistic**: No locks are held in the EJB container or database during a transaction.
- **Verify columns.** <verify-columns> Specifies the columns in a table that should be checked for validity when optimistic concurrency strategy is used. Choose from the list:
  - **Read**: Checks all of the columns in the table that are read during the transaction.
  - **Modified**: Checks only the columns that are updated by the current transaction.
  - **Version**: Indicates that a version column exists in the table. This column is used to check validity.
  - **Timestamp**: Indicates that a pseudo timestamp column exists in the table.This column is used to check validity.
- **Optimistic column. <**optimistic-column**>** The database column that contains a version or timestamp value used for optimistic concurrency:

The **Automatic key** subtab enables you to use the *Automatic Sequence/Key Generation* facility. You can set the following parameters:

- **Generator type.** <generator-type> The type of the key generator.
- **Generator name.** <generator-name> The name of the key generator in accordance with its type.
- **Key cache size.** <key-cache-size> The size of the key cache.

WebLogic 7.0 Inspector tabs for message-driven EJBs

The WebLogic 7.0 Inspector tab for a message-driven EJB gives access to the optional deployment properties associated with messaging. The first two fields, *Max beans in free pool* and *Initial beans in free pool*, are identical to those for WebLogic 6.0/6.1 tab for entity beans. (See "WebLogic 6.0 and WebLogic 6.1 Inspector tabs" on page 324.)

The remaining fields on the WebLogic 7.0 Inspector tab for a message-driven EJB are:

- **Initial context factory.** <initial-context-factory> The initial ContextFactory that the container uses to create its connection factories. If not specified, this defaults to `weblogic.jndi.WLInitialContextFactory`.
- **Provider URL.** <provider-url> The URL provider for the InitialContext. This value is typically host:port.
- **Connection factory JNDI name.** <connection-factory-jndi-name> The JNDI name of the JMS ConnectionFactory that the bean uses for its queues and topics. If not specified, this defaults to `weblogic.jms.MessageDrivenBeanConnectionFactory`, which must be declared in config.xml.
- **JMS pooling interval.** <jms-polling-interval-seconds> The number of seconds between each attempt to reconnect to the JMS destination.
- **JMS client id.** <jms-client-id> The client id for a topic connection with a durable subscription.

Working with EJBs

This section describes the details of working with entity EJBs, session EJBs, and message-driven EJBs.

Entity beans

When you create an entity bean from the diagram toolbar button or the Object Gallery, Together generates skeletons for the implementation bean, the remote and home interfaces, and a primary key.

The entity bean Inspector and the Designer are helpful for modifying the code for an entity bean. We discuss the Inspector and the Designer here.

Default entity bean code

The default code for an entity bean consists of the following:

- Entity bean implementation class with:
  - `EntityContext` attribute

- One `int` type field that serves as a primary key
- Required method declarations:

  `setEntityContext` and `unsetEntityContext`

  `ejbActivate` and `ejbPassivate`

  `ejbCreate`, `ejbPostCreate`, and `ejbRemove`

  `ejbStore` and `ejbLoad`

  `ejbFindByPrimaryKey`

  `getField` and `setField`

- EJB home interface with two method signatures: `ejbCreate` and `findByPrimaryKey`
- EJB remote interface with two method signatures: `getField` and `setField`
- EJB primary key class with:
  - Default field (`int` type)
  - Primary key, hash code, and equals methods

Class diagrams with entity EJB components show dependency links from the displayed interfaces and primary key class to the implementation (bean) class.

When you create an entity bean, Together generates a primary key (`int field1`) and stubs for the general EJB methods such as `ejbCreate()`. Comments in the code indicate where you should provide the actual definitions: `// Write your code here.`

Note The top command on the right-click menu of a property, create method, business method, primary key, or field creates a new item of the same kind. For example, the top command on the right-click menu for a business method is **New Business Method.**

Entity beans that satisfy the EJB 2.0 specification can have local interfaces. For instructions on generating local interfaces, see "Creating EJB components from the class diagram" on page 312.

Setting the persistence

Entity bean persistence can be bean managed (BMP) or container managed (CMP). Together generates BMP entity beans by default. To create a CMP bean, you can first create a BMP entity bean and then switch the persistence to CMP.

To switch the persistence of an entity bean from BMP to CMP:

1. Open the entity bean Inspector by right-clicking the bean in the diagram and choosing **Properties**.
2. Choose the **Entity EJB** tab in the Inspector.

3. Choose the **General** tab at the bottom of the Inspector.
4. In the *Persistence management* setting, click the *Container managed* radio button. See Figure 72 as an example.
5. Click **Yes** in answer to the question: "All ejbFind methods will be deleted, continue?" CMP bean do not implement finder methods.

Creating primary keys

Any field that is serializable is a potential primary key.

To define a simple primary key:

1. Open the EJB Inspector (right click the bean and choose **Properties**).
2. Go to the **Fields** tab of the Inspector.
3. Choose a field with a serialized type. Then check its *Primary key* column. Make sure the *Primary key* column for any other field is clear.
4. Go to the **General** tab and check *Simple primary key.*

Deploying an entity bean requires a database and a table with fields corresponding to the bean.

Tip You can create a database from entity beans. Choose **Tools | Generate | DDL** on the main menu to open the Generate DDL Expert. In *Export from*, choose *Enterprise JavaBeans.* Choose the diagram name, and click **Next.** On the next page, check *Run*, and click **Add** to open an additional page. Enter settings in the connection profile. Test the connection with the database.

Creating container-managed relationships

The EJB 2.0 specification provides for container-managed relationships (CMRs) between CMP entity beans that have local interfaces.

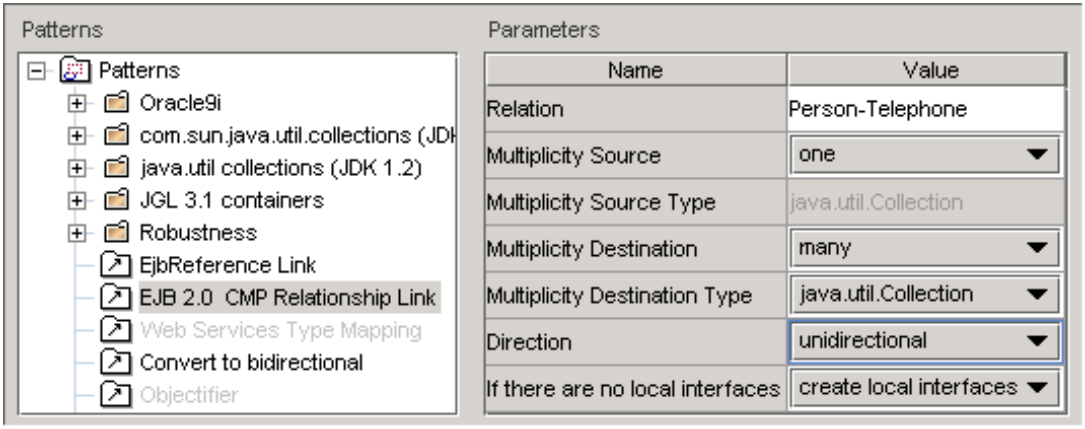
To create a container-managed relationship in a project satisfying EJB 2.0:

1. Open the class diagram containing the two CMP entity beans.
2. Click the **Association by Pattern** button on the toolbar. Connect the two CMP beans with the relationship.
3. In the left pane of the resulting dialog box, click *EJB 2.0 CMP Relationship Link.*
4. Fill in CMR parameters on the right pane.
5. Click **Ok** to finish the process and quit the dialog.

Figure 74 shows a unidirectional one-to-many CMR named `Person-Telephone`. This dialog setting specifies creating local interfaces if they do not exist rather converting remote interfaces.

The class diagram displays a new virtual field for the CMR on the source bean. It also displays a virtual association connecting the two beans, labeled with the relation name and the multiplicities. Together stores CMR properties as Javadoc comments in the code of the source bean.

Figure 74 Applying a CMR pattern



A many-to-many CMR assumes that there is an "intermediate linking table" that describes the many-to-many mapping. The table name as well as other properties of the CMR are listed in the CMR Inspector.

To change the properties of a CMR:

1. Right-click the virtual field member for the CMR in the diagram and choose **Properties**.
2. Choose the **EJB relationship** tab.
3. Change the properties as needed, including the *Relationship table name* to specify the name of the intermediate table.

Session beans

This section discusses how to modify the code for session beans. You can use the right-click menu of the bean or the session EJB Inspector to tailor much of the code. You must use the JBuilder Editor to fill in some of the details such as business logic.

Default session bean code

When you create a session bean using the diagram toolbar or the Object Gallery, Together generates skeletons for the implementation class and remote and home interfaces. The default code consists of the following:

- Session bean implementation class with:
  - Session `context` attribute

    - `setSessionContext` method
    - `ejbActivate` and `ejbPassivate` methods
    - `ejbCreate` and `ejbRemove` methods
- EJB home interface that has a `create` method with no parameters
- EJB remote interface

The default code does not include business methods or additional creators. It also does not have local interfaces, which are possible if the project's application server conforms to EJB 2.0 specifications.

To create new business methods, creators, or local interfaces:

1. Right click the session bean in the diagram to open its right-click menu.
2. *Business method:* Choose **New | Business method.** Edit the name of the method using the in-place editor on the diagram.

   Together writes the method in the implementation bean and puts its signature in the remote interface. Use the JBuilder Editor to put business logic in the body of the method.

Note You can also use the session EJB Inspector to create new business methods.

3. *Creator:* Choose **New | Create method.** Edit the name of the creator using the in-place editor on the diagram.
4. *Local interfaces:* Choose **New | Create local interfaces**. Together gives the new local interfaces the same method signatures as their non-local counterparts.

Making the session EJB stateful or stateless

A stateful session bean can retain its state on behalf of an individual client. A stateful bean has three features:

- A serializable member variable for the conversational state
- A method `ejbcreate(<parameter>)` that has an input parameter
- Notation in the deployment descriptor to indicate the bean is stateful

You can use the JBuilder Editor or the right-click menu of the bean to create the serializable member variable and the method. Together writes the deployment descriptor when you deploy the session bean.

To make the session bean stateful or stateless:

1. Right click the session bean in the diagram and choose **Properties**.
2. Choose the **Session EJB** tab at the top. Then choose the **General** tab at the bottom of the Inspector.

3. At the bottom of the **General** tab, check the *Stateful* box if you want a stateful bean and clear it if you want a stateless bean.

Javadoc comments corresponding to statefulness appear in the source Java code.

Message-driven beans

This section discusses how to modify the code for message-driven beans. You can use the right-click menu of the bean or the message-driven EJB Inspector to tailor much of the code. You must use the Together Editor for some details such as the onMessage content. You must use the Together Editor for some details such as the onMessage content.

Setting the project properties for message-driven beans

A project containing message-driven EJBs must satisfy two requirements:

- The application server for the project is Generic 2.0 or EJB 2.0 compliant.
- The Search/Classpath libraries include `j2ee.jar`.

To specify the appropriate Search/Classpath for a project with message-driven beans:

1. Choose **Project | Project Properties** from the main menu.
2. Go to the **Search /Classpath** tab. (Click **Advanced** if the tab is not visible.)
3. Clear the checkboxes *Include standard libraries* and *Include Classpath.* They are not necessary.
4. Click **Add Path, Library, or Archive.**
5. In the resulting file chooser dialog, navigate to `j2ee.jar` and click **Select**. You can find `j2ee.jar` in `$TGH$/bundled/j2ee/lib`.
6. Click **Ok** to complete the task.

Default message-driven bean code

When you create a message-driven EJB using the diagram toolbar or the Object Gallery, Together generates the bean skeleton. There are no interfaces. The default code consists of the following:

- `MessageDriven` bean context attribute
- `setMessageDrivenContext` method
- `ejbCreate` and `ejbRemove` methods
- `ejbActivate` and `ejbPassivate` methods
- `onMessage` method

The message-driven EJB Inspector is useful for modifying many properties of message-driven beans. It contains only two mandatory subtabs: *References* and *Deployment properties.*

The References tab is almost the same as session and entity EJBs except that there are no EJB security role references. The meanings of the other five types are the same as for session and entity beans. More details are in the section "References tab" on page 317.

Setting deployment properties for message-driven beans

Deployment properties is a tab on the message-driven EJB Inspector. The special message-driven bean properties include:

- **JNDI name to bind to:** Specifies the JNDI name of the JMS ConnectionFactory that the message-driven bean accesses to create its queues and topics.
- **Acknowledge node:** Possible values are *Auto-acknowledge* and *Dups-ok-acknowledge*. This is used for beans with bean-managed transaction management.
- **Destination type:** Possible values are `javax.jms.Queue` and `javax.jms.Topic`.
- **Destination JNDI name:** JNDI name of the queue or topic.
- **Subscription durability:** Possible values are *Durable* and *NonDurable.*
- **Message selector:** A string to filter the incoming messages.
- **Security identity:** Possible values are *none, user-caller-identity,* and *run-as-specified-identity.*

For a discussion of deployment properties that apply to all EJB types, see the section "EJB Inspector tabs" on page 315.

Sharing home and remote interfaces

Two or more entity beans or two or more session beans can share the same set of home and remote interfaces. This is potentially useful in deploying differing implementations of the same interfaces on different servers.

Creating a second implementation class

Before attempting to create a second implementation class, you should develop one complete EJB component. Then you can create a second class, refactor it into an EJB, and specify the home and remote interfaces of the first bean as those of the new one.

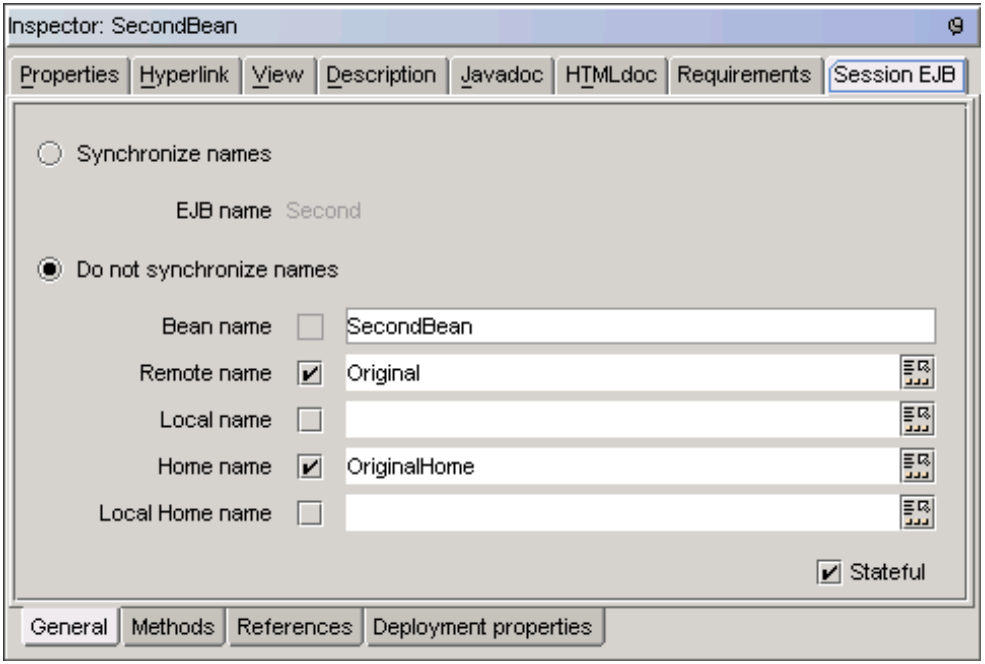
To create a second implementation class:

1. Create a new class in an existing class diagram or a new one.

2. Refactor the new class into an EJB:
    1. Right click the class node in the diagram and choose **Choose Pattern**.
    2. In the resulting dialog box, expand *EJB* and then choose *Entity EJB* or *Session EJB* in the left pane. Your choice should match the type of the first bean.
3. Right click the new bean and choose **Properties**.
4. In the Inspector for the new bean, click the **Entity EJB** or **Session EJB** tab on the top (depending on the type of bean). Click the **General** tab on the bottom.
5. Choose **Do not synchronize names.**
6. In the *Remote name* field, click the browse button .
7. In the resulting dialog, expand the *Model* node of the tree-view. Locate and choose the remote interface belonging to the first implementation (that is, the remote interface that you want the new bean to use).
8. Click **Ok** to accept the selection. Click **Change class** in response to the prompt.
9. Follow the same procedure in the *Home name* field, selecting the home interface for the first bean in the resulting dialog.

Figure 75 shows the Inspector for a `SecondBean`, which shares remote and home interfaces with `OriginalBean`.

Figure 75 Sharing remote and home interfaces



To correct business methods exposed from shared interfaces, uncheck **Synchronize remote interface** methods in the options dialog (Choose **Tools | Options | <level>** from the main menu, and expose the *EJB* node on the left).

Deleting implementation classes with shared interfaces

Normally when you delete an EJB implementation class from a diagram, Together deletes the home and remote interfaces as well. This includes the source files and their visual diagram elements.

In the case of shared interfaces, deleting elements from the diagram does not automatically result in deletion of the corresponding source code files. Instead, Together deletes only the visual diagram elements. In order to delete the source files, you must explicitly delete implementation classes and home and remote interfaces. You can do this by selecting them in the **Model** tab of the Explorer and choosing **Delete** from the right-click menu.

Customizing the default EJB code

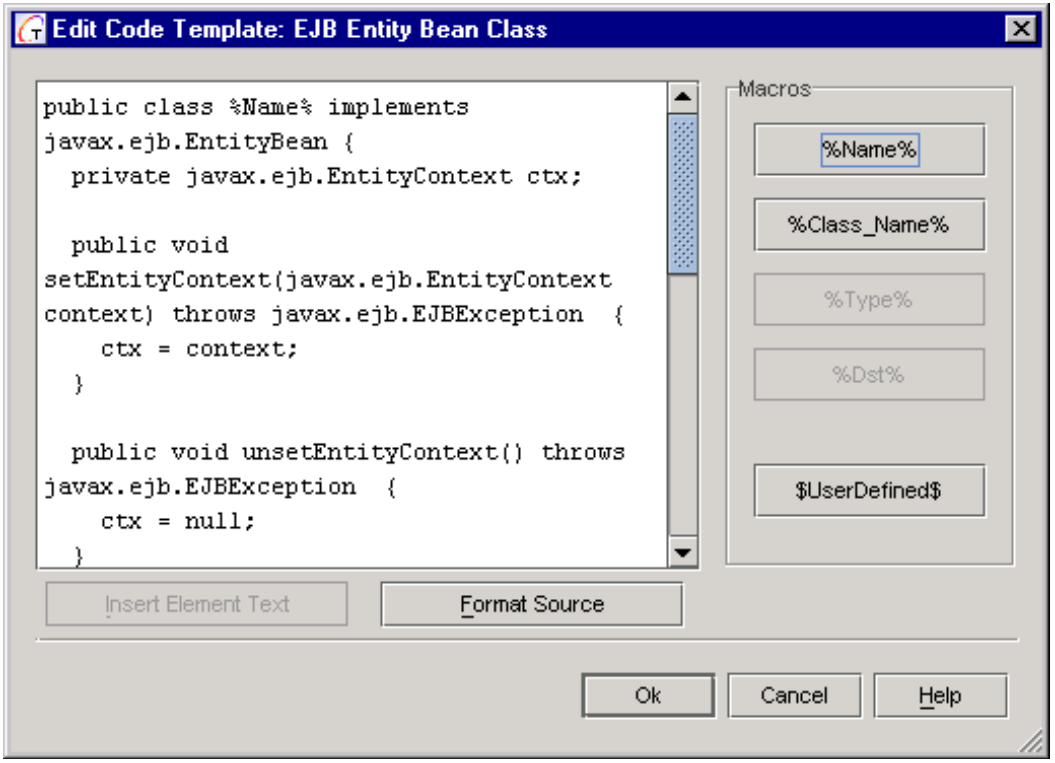
You can customize the skeletons that Together generates when you create an EJB by editing their code templates. There are templates for entity bean, session bean, and message-driven bean classes, primary key class, and home, remote, local home, and local interfaces.

To edit an EJB code template:

1. Choose **Tools | Code Template Expert** from the main menu.
2. On the resulting dialog box, choose *Java* as the *Template Language* and *Class* as the *Template Category.* Click **Next** to continue.
3. In the list of template folders on the left pane, expand *EJB10, EJB11,* or *EJB20* to edit EJB code satisfying the EJB 1.0, EJB 1.1, or EJB 2.0 specifications respectively. Choose the template to edit from the items shown and click **Next** to continue.
4. Click **Edit Template Code** to edit the template.
5. Edit the code and click **Ok** when you finish.

Figure 76 shows the dialog box for editing the code for an entity EJB satisfying EJB 2.0 specifications.

Figure 76 Editing the entity bean class template with the Code Template Expert



Chapter

21

Assembling EJB Modules

Many multi-tier distributed applications use Enterprise JavaBeans (EJBs) to implement business logic. Together provides a means of visually modeling the assembly of EJB components into EJB modules, which can be subsequently archived into JAR files and deployed directly to application servers.

This chapter discusses assembling EJB modules on special EJB assembler diagrams. The chapter includes the following topics:

- "Visual assembly of EJB applications" on page 337
- "Creating and modifying EJB modules" on page 338
- "Displaying EJB shortcuts on EJB assembler diagrams" on page 340
- "EJB verification and correction" on page 343
- "Deploying an EJB module" on page 344

Visual assembly of EJB applications

Together provides special *EJB assembler diagrams* for modeling EJB applications. An EJB assembler diagram represents a single EJB module. An EJB module is a logical grouping of EJBs along with a corresponding deployment descriptor.

Note The Together EJB assembler diagram is a visual equivalent of the *Application Assembler* described by the EJB 2.0 Specification. The Together J2EE Deployment Expert corresponds to the *Application Deployer.* Refer to page 432 of the EJB 2.0 Specification for details.

An EJB assembler diagram enables you to group several EJBs in a JAR and deploy it to an application server.

Creating and modifying EJB modules

Creating an EJB module in Together is the same as creating an EJB assembler diagram. This section discusses creating EJB assembler diagrams and their elements.

Creating an EJB module

You can create an EJB assembler diagram (EJB module) using the Object Gallery.

To create an EJB assembler diagram:

1. Choose **File | New** on the main menu to open the Object Gallery.
2. On the left pane, choose *Enterprise.*
3. On the right pane, click *EJB Module*.
4. Click **Next.**

   The resulting dialog box is an expert that guides you through creating and populating an EJB module.
5. Fill in the text fields.
   - **EJB module name.** The name of the new EJB assembler diagram.
   - **Display name.** The EJB module name for the deployment descriptor.
   - **EJB module location:** The folder for the module.
6. Click **Add from Project** to open the *Select EJBs* dialog.
7. On the left pane, open each folder containing EJBs. CTRL+click each bean class that you want to add to the module.
8. Click **Add** and then click **Ok** to close the dialog.
9. Repeat the same process to add application clients to the EJB module.
10. Click **Finish**.

The resulting EJB assembler diagram is an EJB module that contains shortcuts to all of the EJBs that you selected.

Note An EJB assembler diagram contains only EJB shortcuts. The actual EJBs belong on class diagrams instead of EJB assembler diagrams.

Placing elements on an EJB assembler diagram

The items on an EJB assembler diagram include shortcuts to EJBs, references, method permissions, security roles, principles, and links. EJB assembler diagrams enable you to refine the architecture of an EJB application by linking appropriate diagram elements together. For example, you can link method permissions to security roles or EJBs to resources.

The Designer toolbar at the left side of the diagram has buttons for most of the key EJB assembler elements. Table 40 lists the special buttons.

Table 40 Special EJB assembler diagram design elements

Button	Description
	EJB Shortcuts: Adds or removes shortcuts to EJBs.
	Security Role: Defines security roles for EJB clients. (See Chapter 28, "J2EE Platform Security Support" on page 431.)
	Principal: Creates a user or a group of users separate from their security roles.
	Method Permission: Shows the required permissions on one or more methods.
	EJB Properties: Creates a separate EJB properties element. (Use this to represent EJB deployment properties separate from the EJB or for changing these properties.)
	Container Transaction: Defines transaction attributes for the methods of EJB home and remote interfaces. (See Chapter 22, "Container Transactions and EJB References" on page 347.)
	EJB Reference: Creates a reference to a corresponding EJB. Defines a remote type for EJB reference: EJB, or EJB local (for EJB 2.0 specification).
	Environment Reference: Creates an environment reference.
	Resource Reference: Creates a reference to a resource or resource-environment (for EJB 2.0 specification).
	Bundled Files: Creates a set of bundled files (such as Web files).

You can access the properties of any element on an EJB assembler diagram by using its inspector. The inspectors vary according to the type of element.

Setting properties of an EJB assembler diagram

You can use the EJB assembler diagram inspector to set the diagram properties. To open the inspector, right click on the diagram background and choose **Properties.**

The tabs in the inspector for an EJB assembler diagram include the ordinary class diagram tabs as well as an **EJB** tab. The **EJB** tab lists properties used during the deployment process:

- **Module name.** Name of an archive `*.jar` file that is created during the deployment process. By default this name is the same as the *Name* on the **Properties** tab.
- **Display name, Small icon, Large icon.** Optional parameters for the deployment descriptor.

Displaying EJB shortcuts on EJB assembler diagrams

EJB assembler diagrams contain shortcuts to EJBs. You can place shortcuts on an EJB assembler diagram when you first create it using the Object Gallery. You can also add them to the diagram later by using the toolbar or the diagram right-click menu.

Creating EJB shortcuts

A shortcut is a node for an element that resides somewhere else in the project.

To create an EJB shortcut on an EJB assembler diagram:

1. Open the EJB assembler diagram.
2. To include EJBs that are not part of the current project, specify the paths to them in the **Search/Classpath** tab of the Project Properties dialog (**Project | Properties**).
3. Click the **EJB Shortcuts** button from the toolbar. This opens a dialog for selecting shortcuts.
4. In the dialog, expand the *Model* node and locate the EJB classes and interfaces from the project that you want to display in the diagram. CTRL+click the desired classes and interfaces on the left and click the **Add** button.
5. In the dialog, expand the *Search/Classpath* node and locate the EJB classes and interfaces from outside your project (if any) that you want to display in the diagram. CTRL+click them on the left and click the **Add** button.
6. Click **Ok** to add the selected EJB shortcuts on the diagram.

The EJB assembler diagram displays the shortcuts to the items that you selected. The shortcuts retain all of the characteristics of the actual items. For example, if the original item has a hyperlink, the item in the corresponding shortcut also has the hyperlink.

If you modify a shortcut on the diagram, the modifications apply to the actual bean as well with one exception. Deleting a shortcut removes it from the diagram; deleting the shortcut does not remove the actual bean.

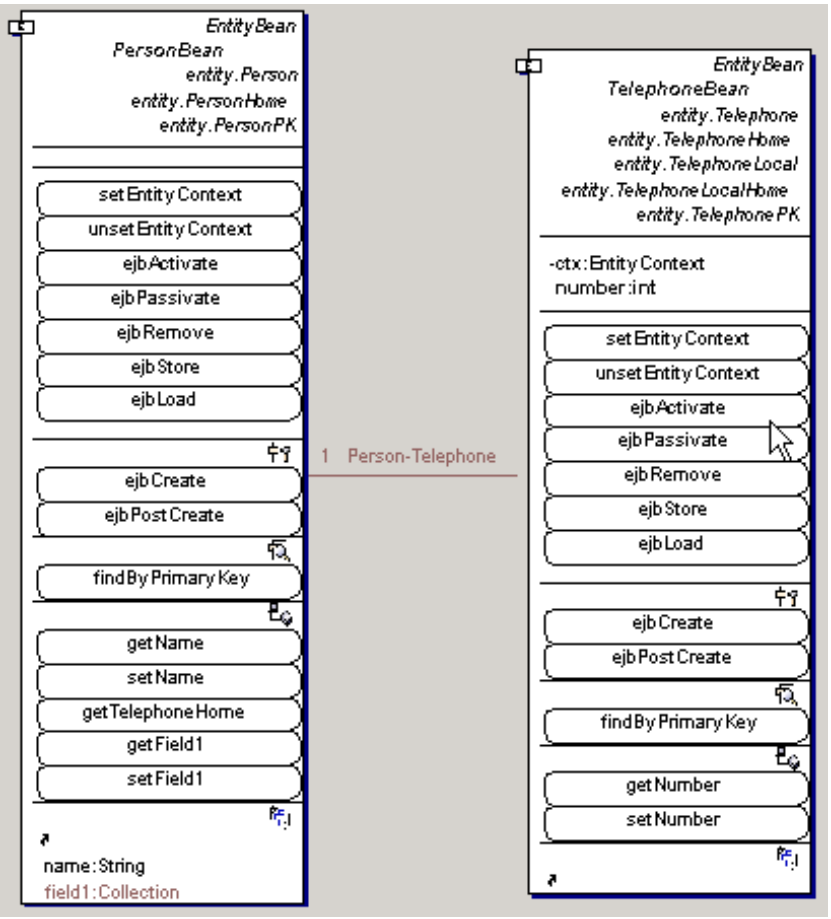
Displaying EJBs on EJB assembler diagrams

The display of an EJB shortcut on an EJB assembler diagram is similar to the display of an EJB component on a class diagram. The main visual distinction is that in an EJB assembler diagram, the methods have elliptical borders.

Figure 77 shows shortcuts for two entity nodes on an EJB assembler diagram with the following features:

- A container-managed relationship named `Person-Telephone` links the nodes.
- The `PersonBean` has hidden attributes. (To hide attributes, right click the bean shortcut and choose **Hide | Attributes.**)
- The fields for the `PersonBean` show at the bottom of the node. (This is the result of hiding the attributes.)

Figure 77 EJB shortcuts on an EJB assembler diagram.



Showing and hiding EJB component parts

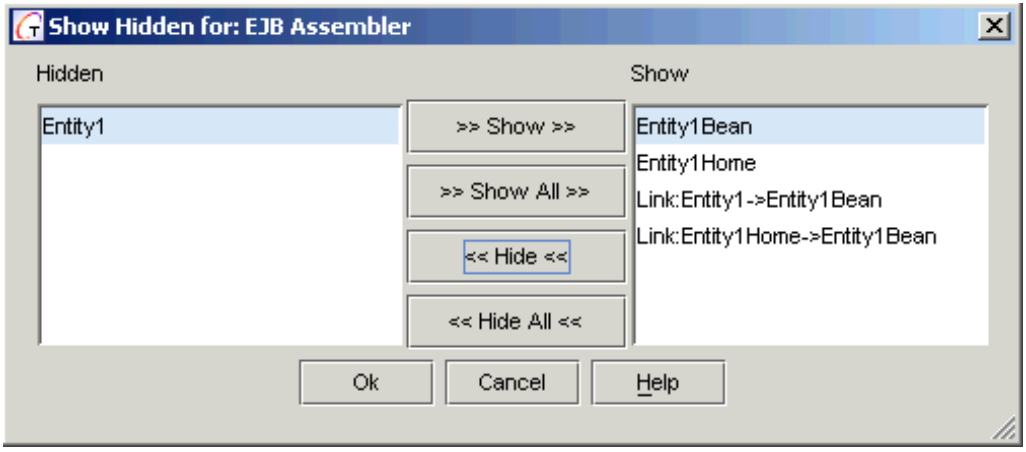
When you create EJB shortcuts, you can choose interfaces or primary key classes as shortcuts. If they do not appear on the EJB assembler diagram, the *View Management* option settings are hiding them. For instructions on how to change those settings, see "Setting component editing and display options" on page 308.

You can hide or show individual elements on the diagram using the **Show Hidden** command from the right-click menu.

To show or hide elements on an EJB assembler diagram:

1. Right-click the diagram background and choose **Show Hidden.**
2. In the resulting dialog box (shown in Figure 78), choose the hidden elements on the left that you want to see in the diagram. Then click **Show.**
3. Choose the shown elements on the right that you want to hide and click **Hide.** Click **Ok** to quit the dialog and change the diagram.

Figure 78 Show or hide elements on a EJB assembler diagram.



How to copy and paste EJB shortcuts

To create a copy of an EJB shortcut on an EJB assembler or Web application diagram:

1. Open the EJB assembler diagram that contains the shortcut you want to copy.
2. Right click the EJB shortcut and choose **Copy**.
3. Open the diagram where you want to place the copy.
4. Right click the diagram and choose **Paste.**

If you paste the shortcut to a different diagram, the new shortcut has the same name as the original. If you paste the shortcut to the same diagram, the copy has a different name from the original. In that case, Together creates the bean class corresponding to the new shortcut on the class diagram.

If you want to cut a shortcut and then paste it on another diagram, right-click the shortcut and choose **Cut.** Click on the selected diagram, and then choose the **Paste** (or **Paste Shortcut**) option from the options of the right-click menu. The mechanism of copying is the same: the objects will be created.

Note A single diagram can contain only one shortcut to a particular object.

EJB verification and correction

Entity and session EJBs must conform to EJB 1.0, EJB 1.1, or EJB 2.0 specifications. Message-driven EJBs must conform to EJB 2.0 specification only. Descriptions of these specifications are beyond the scope of this document and are available on the Sun website at *http://www.java. sun.com*.

Verification overview

Together provides verification for the EJB components in a project including:

- Conformance to the common requirements for all specifications: the correctness of inheritance, home and remote interfaces, testing the primary key type, and so on.
- Conformance to the specific requirements of the project's target application server (EJB 1.0, 1.1, or 2.0). EJB 2.0 verification includes checking conformance for local interfaces and message-driven beans.

Note The syntactic rules for verification are defined in configuration files under `$TGH/modules/com/togethersoft/modules/ejbbase/impl/verification/config`. The rules are organized in three folders corresponding to the EJB specifications. You can add new requirements or create customized syntactic rules by editing the appropriate configuration files.

Using verification and correction

You can verify EJBs from class diagrams or from EJB assembler diagrams. When you verify an EJB, the Message pane displays the results in an EJB Verification tab.

The **EJB Verification** tab displays error messages that are colored blue or red:

- Blue error messages report errors that Together cannot fix.
- Red error messages report errors that Together can fix.

Double clicking an error message, and switching focus to JBuilder highlights the place the error was detected in the JBuilder Editor.

To verify a single EJB:

1. Right-click on the EJB in the diagram and choose **Verify EJB**.
2. Open the **EJB Verification** tab in the Message pane.
3. Together displays messages for any errors it detected according to the current project specification. Together does not automatically correct the errors.

To correct the errors:

1. Right click one of the error messages colored red.
2. Choose **Correct All.** Together corrects the code and turns the color of the message to blue.

You can use the verification and correction process during deployment of EJB modules. (See Chapter 27, "J2EE Deployment" on page 399.)

Deploying an EJB module

Together provides an expert dialog that simplifies the process of deploying EJB modules. You can run the J2EE Deployment Expert from a class diagram that contains an EJB component or an EJB assembler diagram.

Set deployment properties in the *EJB* tab of the diagram Inspector.

Note If the target application server for your project is Borland Enterprise Server 5.2, the inspector for an EJB module displays the **Borland Enterprise Server 5.2** tab. Use this tab for setting additional deployment properties. For more detail information on the contents of the Borland Enterprise Server 5.2 tags, see **http://info.borland.com/techpubs/books/bes/pdfs52/.**

To invoke the J2EE Deployment Expert:

1. Open a class diagram containing an EJB or an EJB assembler diagram.
2. From the main menu, choose **Deploy | J2EE Deployment Expert.**

The difference between deploying from a class diagram and deploying from an EJB assembler diagram is as follows:

- **Class diagram:** For "fast track" deployment with default values or deployment prototyping.
- **EJB assembler diagram:** For "full featured assembly information" with control over security and permissions.

Important If you try to deploy the same EJB component to the same application server twice, you will get an exception saying that the application server cannot identify beans with the same names. To deploy an EJB a second time from a different EJB assembler diagram, set different deployment parameters (JNDI and the bean name) for the EJB shortcuts.

For information on using the J2EE Deployment Expert, see Chapter 27, “J2EE Deployment” on page 399.

Chapter

22

Container Transactions and EJB References

The deployment descriptor for an EJB module determines the transaction contexts for EJB methods. The deployment descriptor also determines how EJBs can communicate within the container and gain access to required resources.

This chapter discusses Together's support for both of these concepts. The chapter includes the following topics:

- "Container transaction elements" on page 347
- "EJB references" on page 350

Container transaction elements

Container transactions are the indivisible interactions between containers and databases. Deployment descriptors define transaction behaviors of the methods of an EJB. Their `container-transaction` elements specify *transaction attributes.* Together supports transaction attributes through *container transaction* diagram elements.

Container transaction attribute types

There are six different kinds of container transaction attributes for EJB methods:

- **Required:** (`TX_REQUIRED`) Executes in the context of a transaction. The container uses the current transaction context if one exists. Otherwise it starts a new one.
- **RequiresNew:** (`TX_REQUIRES_NEW`) Executes in the context of a new transaction. The container suspends any transaction already in progress until the new one is completed.

- **NotSupported**: (`TX_NOT_SUPPORTED`) Executes outside the context of any transaction. The container suspends any transaction in progress until the method completes.
- **Supports:** (`TX_SUPPORTS`) Executes inside the current transaction if one exists. If there is not a transaction in progress, the method executes without a new transaction.
- **Mandatory:** (`TX_MANDATORY`) Executes inside the current transaction. If there is no transaction, the container throws a `javax.transaction.TransactionRequiredException`.
- **Never:** (`TX_NEVER`) Executes outside the context of any transaction. If the client provides a transaction context, the container throws a `java.rmi.RemoteException`.

A transaction attribute can apply to all the methods of an EJB or it can apply to an individual method.

Creating container transaction elements

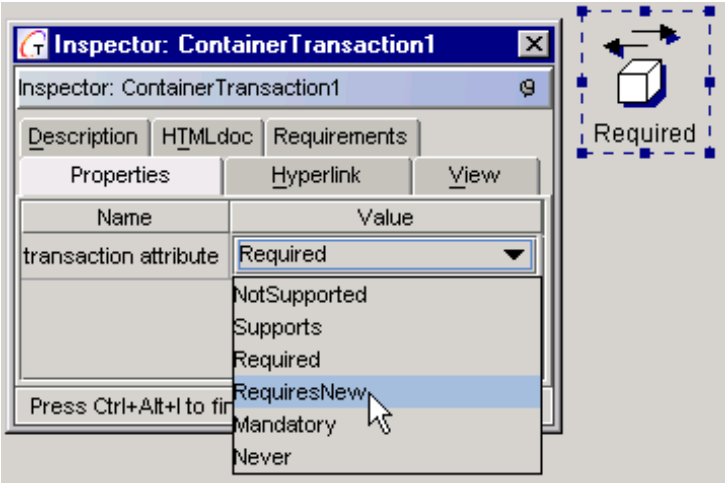
Container transaction diagram elements reside on EJB assembler diagrams.

To create a container transaction element:

1. Open the EJB assembler diagram for the EJB module that you are building.
2. Create a shortcut to the EJB that you want to use. (See "Creating EJB shortcuts" on page 340.)
3. Click **Container Transaction** on the diagram toolbar and click the diagram. This creates a container transaction element as shown in Figure 79.
4. You can set the value for the transaction attribute through the container transaction element inspector. To access its inspector, right click the element and choose **Properties.**
5. The default value of *transaction attribute* is *Required*. Use the dropdown list to set the value as desired, as illustrated in Figure 79.

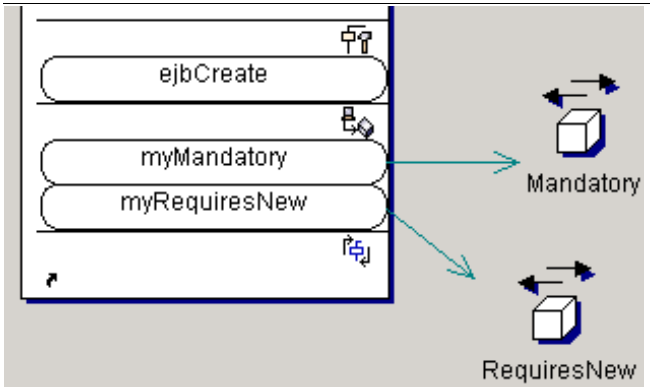
Repeat this process to create as many container transactions as needed for the different EJB methods.

Figure 79 Container transaction element inspector



To associate a container transaction element with a method or with an EJB, draw a link from method or the EJB to the container transaction element. Figure 80 shows links from the methods in a session EJB to two container transactions.

Figure 80 Linking methods to container transactions



If there is a link from an EJB to a container transaction element, then the container transaction element type is the default type for all methods in the EJB. You can override the default for a particular method in the EJB by linking the method to a different type container transaction element.

During deployment, Together generates all necessary descriptions connected with container transaction diagram elements and the corresponding methods for the deployment descriptor. The container transaction elements are added to the JAR of the EJB module (the EJB assembler diagram).

EJB references

One EJB cannot directly call the methods of another. Such bean-to-bean communication requires an EJB reference. References are useful for accessing resources, security, and environment variables as well.

Together provides support for EJB references with two different constructs: *reference attributes* in EJB implementation classes, and *EJB reference elements* on EJB assembler, Web application, and application client diagrams.

Both reference attributes and EJB reference elements translate into deployment descriptor tags. When you initiate the deployment process from an EJB assembler diagram, Together adds all necessary descriptions connected with EJB references to the deployment descriptor (`*.xml` file) and adds EJB reference elements to the JAR of the EJB module. (This is true as well if the J2EE Deployment Expert starts from Web application, enterprise application, or application client diagrams that contain EJB reference elements. Together adds the EJB references to the WAR, EAR, and application client deployment descriptors.)

EJB reference diagram elements

EJB reference elements enable you to create applications with different combinations of components. A client EJB can get the properties of a referenced EJB through this intermediate EJB reference element. The two EJBs can be in the same JAR file or in different JAR files in the same J2EE module. You can expand the communication among EJBs by creating additional EJB reference diagram elements and redirecting links among EJBs.

An EJB reference element with its own properties provides a flexible connection unit for establishing references between EJBs. You can use the same EJB in different applications without changes. Rather than change the EJB, you merely modify the properties through the corresponding EJB reference element. Indeed, it is possible to create EJB references on a J2EE diagram without having any EJBs in the project.

When you develop a Web application, you can create EJB reference elements instead of creating shortcuts of the actual EJB components to the diagram. Specify properties (home, remote interfaces, and others) in the inspector, and draw a link from the servlet to the EJB reference. This information becomes part of the deployment descriptor.

EJB properties elements

The *EJB properties* element on an EJB assembler diagram enables a client EJB to access the deployment properties of a supplier EJB without specific knowledge of the supplier. You can define EJB deployment properties manually, and link the EJB Properties element to the corresponding shortcut to the EJB in the diagram.

To create an EJB properties element on an EJB assembler diagram:

1. Open the EJB assembler diagram and create a shortcut to the EJB whose deployment properties you want to represent.
2. Click **EJB Properties** on the diagram toolbar, then click the diagram.
3. Draw a link from the EJB shortcut to the new deployment properties element.
4. Open the Inspector of the new element (right click and choose **Properties**).
5. Enter the desired values for EJB deployment properties.

Note The fields in the EJB properties element inspector vary according to the EJB that links to it.

If the EJB properties element links to a CMP entity EJB, the inspector shows database connection properties. You can change the database pool name and the name of the table corresponding to the CMP bean.

The EJB properties element provides the flexibility of using the same EJB in different enterprise or Web applications without modifying the EJB code. It is not necessary to change deployment properties such as JNDI names inside the bean. Instead, you can simply create an additional deployment properties element.

You can use the deployment properties element to refer to this EJB at a later time. For example, you can place a shortcut to the deployment properties element on another EJB assembler or Web application diagram. This effectively includes the corresponding EJB in the other EJB module or Web application.

Note When Together deploys an EJB that is linked to a deployment properties element, the information in the deployment properties element overrides the corresponding information in the EJB.

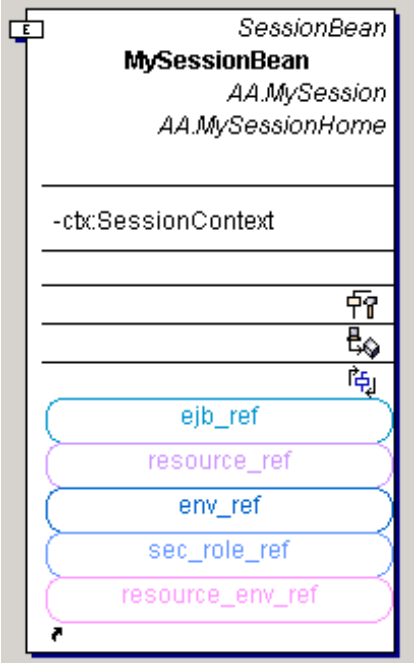
Reference attributes in the client

You can put a reference attribute in the implementation class of the EJB client.

To create a reference attribute in an EJB client class:

1. Right click the EJB on the class diagram (or on a shortcut to the EJB on an EJB assembler diagram) and choose **New | <reference type>.** Figure 81 shows different types of reference attributes with different colors, which display in the bottom compartment of the implementation class node.
2. The new reference attribute is a `String` by default. You can change its name and type with the in-place editor.

Figure 81 EJB references in a shortcut to a session EJB.



Supported EJB reference types

Table 41 displays the types of reference attributes in EJB classes. The right column shows common deployment descriptor tags corresponding to each type.

Table 41 Supported EJB reference attribute types

Reference attribute type	Restriction	Deployment descriptor tag
EJB reference		`<ejb-ref>`
EJB local reference	EJB 2. 0	`<ejb-local-ref>`
EJB resource reference		`<resource-ref>`
EJB resource-environment reference	EJB 2. 0	`<resource-env-ref>`
EJB environment reference		`<env-entry>`
EJB security role reference	Entity or session EJB	`<security-role-ref>`

Reference attribute inspectors

You can adjust the properties of the reference attribute through its inspector (right click the attribute and choose **Properties**). The inspector for each type of reference has a special reference tab (EJB Reference tab, EJB security role tab, and so on) that shows the deployment-critical properties.

The fields on the inspector reference tabs vary according to the reference attribute type as follows:

- Common for all types of reference attributes:
  - **Name:** Name of the reference attribute in the EJB implementation class.
  - **<reference type> name:** Name of the reference in the deployment descriptor (in quotes). This is the same as the initial value of the attribute, which is a `String`.
- EJB reference and EJB local reference:
  - **Remote type:** `local` or `remote`.

Note EJB references and EJB local references differ only in the choice of the *Remote type* field in the reference inspector. Choose `EJB-REF` if you are using a remote interface; choose `EJB-LOCAL-REF` if you are using a local interface.

  - **Reference type**: the type of the supplier EJB (`Entity` or `Session`).
  - **Remote/Home interface:** Remote/home interface of the supplier EJB.
  - **EJB link:** Name of the supplier EJB.
  - **EJB JNDI name:** JNDI of the supplier EJB.
- EJB resource reference and EJB resource environment reference:
  - **Resource type:** `local` or `remote`.
  - **Resource JNDI name:** Actual JNDI name of the resource.
  - **ResourceRefType:** Connection factory type (`javax.sql.DataSource`, `javax.jms.QueueConnectionFactory`, `javax.jms.TopicConnectionFactory`, `javax.sql.XADataSource`,`javax.mail.Session`, `java.net.URL`).
  - **ResourceRefAuth:** Reference authorization (*Container* or *Application*). *Application* is required when the resource type is local.
- EJB environment reference
  - **Environment entry type:** `Boolean`, `Integer`, `String`, `Double`, `Float`, `Byte`, `Short`, or `Long`.
  - **Environment entry value:** Value of the environment variable.
- EJB security role reference (See Chapter 28, "J2EE Platform Security Support" on page 431.)
  - **Role link:** link to security role in the deployment descriptor.

Changing the properties in the inspector modifies the actual EJB code.

EJB reference diagram elements

EJB reference diagram elements on EJB assembler, Web application, and application client diagrams provide the linking between the EJB client and the EJB supplier.

Changing the EJB reference element inspector properties does not modify any code in the EJB component.

To model a client EJB and supplier EJB on in an EJB module:

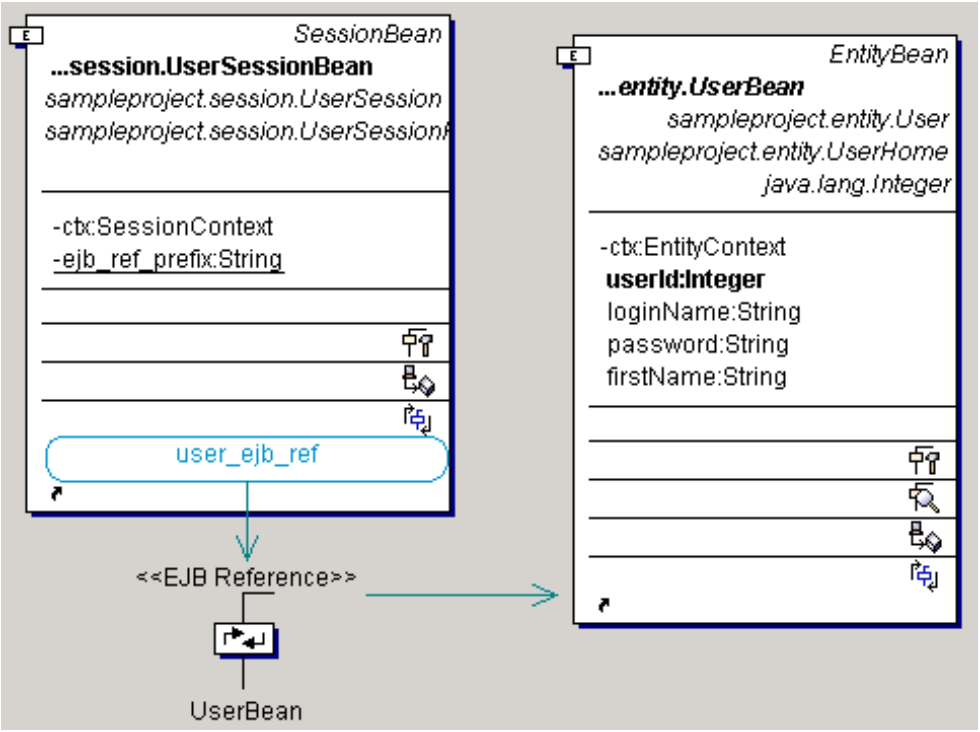
1. Open the EJB assembler diagram corresponding to the EJB module.
2. Create shortcuts to the client and supplier EJBs on the diagram. (Click and choose the client and supplier from the resulting selection manager dialog.)
3. Right click the client EJB and choose **New | EJB reference**. A new reference named `ejb_ref` appears inside the shortcut to the client EJB (also inside the client EJB on the class diagram).
4. Click , then click the diagram. Edit the name of the EJB reference element on the diagram as desired.
5. Draw a link from the client EJB shortcut to the EJB reference element.
6. Draw a link from the EJB reference element to the supplier EJB shortcut.

Figure 82 shows an EJB assembler diagram in which `UserSessionBean` is the client EJB and `UserBean` is the supplier EJB. The client has a reference attribute named `user_ejb_ref`. There is a link from the reference attribute to an EJB reference element, which is also named `UserBean`. There is also a link from the EJB reference element to the supplier.

Note Suppose A, B, and C are two enterprise beans, with the class diagram showing an EJB reference from A to B. Suppose also that there are shortcuts to A and C on an EJB assembler diagram. An intermediate EJB reference element on the EJB assembler diagram can redirect the link from A to C.

If you start the J2EE Deployment Expert with the class diagram in focus, the deployment descriptor makes the reference from A to B. If you start the J2EE Deployment Expert with the EJB assembler diagram in focus, the reference is from A to C instead.

Figure 82 Linking clients to suppliers via references on EJB assembler diagrams



EJB resource and resource environment reference diagram elements

Resource references give EJBs and servlets access databases, mail sessions, JMS objects, and URLs. Environment references are named values that are visible to enterprise applications and Web applications.

You can create a resource reference as an resource attribute or a separate *resource* diagram element. The diagram element has all the properties of the referenced resource (for example, a database object).

To create an EJB resource reference on an EJB assembler diagram:

1. Create a shortcut to the EJB shortcut on the EJB assembler diagram. (Click and choose the EJB from the resulting selection manager dialog.)
2. Right-click on the EJB shortcut and choose **New | EJB resource reference**. This creates a reference attribute named `resource_ref` inside the bean. Right-click the attribute and choose **Properties** to specify its properties. If you are working with a remote interface, choose `remote` as the resource reference type.
3. Click **Resource** on the diagram toolbar, then click the diagram to create a resource element named `Resource1`.

4. Draw a link from `resource_ref` to `Resource1`.
5. Right click `Resource1` and choose **Properties** to open the inspector. You can change the properties of a resource simply by changing the properties associated with the EJB resource design element. You need not make any changes to the properties of the actual EJB.

To create a resource-environment reference, follow the steps outlined above, choosing **New | EJB resource-environment reference** from the right-click menu of the EJB shortcut. The resource type for a resource-environment reference is local rather than remote.

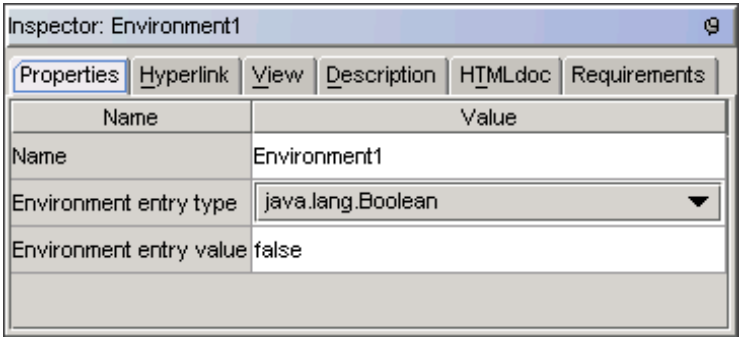
Creating EJB environment references

EJBs can use environment references to access named, static constants. Each environment reference has a type, value, and name. In Together, you can define an environment reference as a reference attribute or as a separate *environment* diagram element with its own properties.

To create an EJB environment reference on an EJB assembler diagram:

1. Create a shortcut to the EJB shortcut on the EJB assembler diagram. (Click and choose the EJB from the resulting selection manager dialog.)
2. Right-click on the shortcut and choose **New | EJB environment reference**. This creates a reference attribute named `env_ref` inside the bean. Right-click the attribute and choose **Properties** to specify its name, type, and value.
3. Click **Environment** on the diagram toolbar, then click the diagram to create an environment element named `Environment1`.
4. Right click `Environment1` and choose **Properties** to open the inspector. Change the values on the EJB environment reference tab. Figure 83 illustrates how to customize a `Boolean` environment entry to `true`.

Figure 83 Environment reference properties



You can link an EJB or an EJB environment reference attribute to an environment element on an EJB assembler diagram. Linking an environment resource element enables you to change the environment properties without making any changes to the code of the EJB.

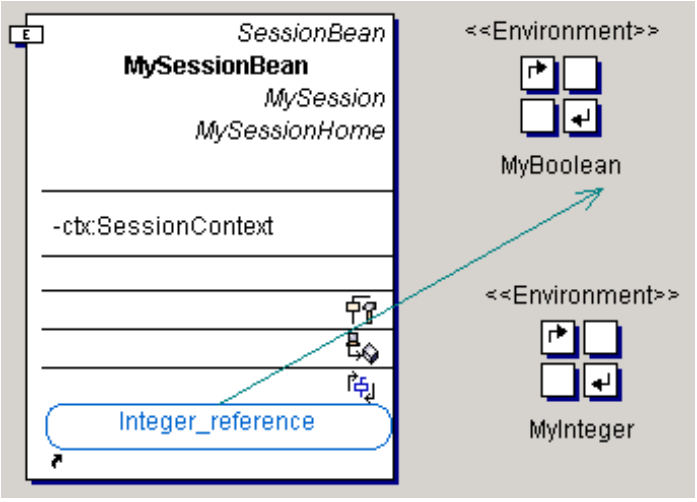
To change the type of the environment variable on an EJB assembler diagram:

1. Add a new environment element to the current EJB assembler diagram.
2. Define its properties through its inspector (right click the environment node in the diagram and choose **Properties**).
3. Draw a link from the environment reference attribute in the EJB to the new environment reference node on the diagram.

Figure 84 shows an environment reference attribute named `Integer_reference` which is linked to an environment reference node named `MyBoolean`. If you start the J2EE Deployment Expert from this diagram, `Integer_reference` translates to the deployment descriptor with `MyBoolean` as a "lookup" name and the type as assigned to `MyBoolean` in its inspector.

Since `MyInteger` is not linked to an EJB reference, it deploys as a separate environment variable.

Figure 84 Environment references



If the properties of the environment reference diagram element and the environment reference attribute in the EJB are different and you start the deployment from the EJB assembler diagram, the J2EE Deployment Expert uses the properties of the diagram element to determine what to write to the deployment descriptor.

Chapter

23

Designing and Developing Web Applications

Together provides a special Web application diagram for creating and building the Web application archives (WAR files) that store all JSPs, servlets, EJBs, and taglibs in a Web application. This chapter discusses the details of using Web application diagrams to design and develop Web applications.
The chapter includes the following topics:

- "Working with Web application diagrams" on page 359
- "Developing applets" on page 366
- "Working with servlets" on page 366
- "Working with JSPs" on page 369
- "Working with tag libraries" on page 370
- "Working with filters" on page 373
- "Working with listeners" on page 375
- "Deploying a Web application" on page 377

Working with Web application diagrams

Together's Web application diagram enables you to model the assembly of JSPs, servlets, filters, and other web files in a Web application. From a Web application diagram, you can generate a deployment descriptor for the application, assemble Web components to a `*.war` archive, and deploy it to an application server.

This section explains how to create Web application diagrams and the basic techniques for working with them.

Creating a Web application diagram

You can create a new Web application diagram from the Designer pane toolbar or by creating a Web application in the Object Gallery.

To create a new Web Application diagram from the Designer toolbar:

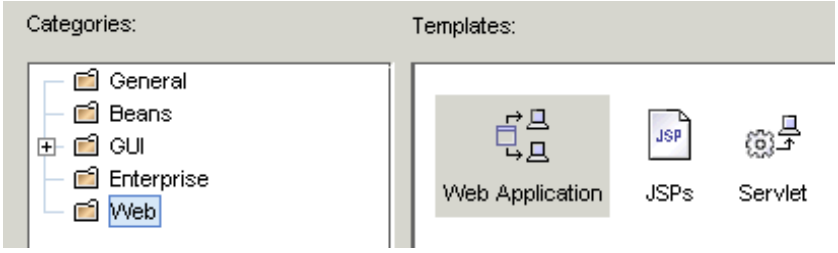
1. Click the **New Diagram** button on the Designer toolbar.
2. In the resulting dialog box, click the Together tab.
3. Choose *Web Application diagram.* Fill in the name of the diagram and its package location.
4. Click **Finish**.

As an alternative, you can use the Object Gallery to create a Web application diagram and populate it with existing elements such as JSPs, servlets, and Web files.

To create a Web Application with the Object Gallery:

1. Choose **File | New** on the main menu. This displays the Object Gallery.
2. Choose *Web* from the categories on the left, then choose *Web Application* from the templates on the right, as shown in Figure 85.

Figure 85 Choosing a Web Application from the Object Gallery



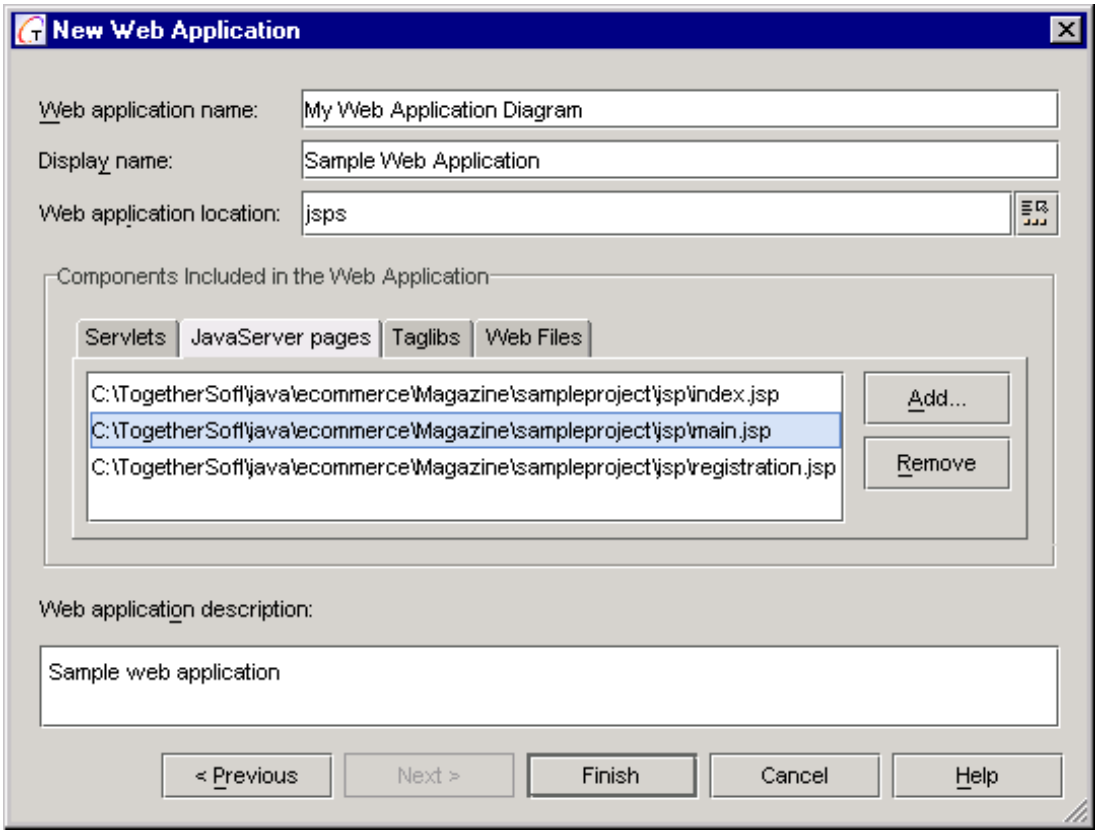
3. Click **Next.** The resulting dialog box is an expert that guides you through the process of creating the Web application, as illustrated in Figure 86.
4. Fill in the New Web Application page in the Expert:
   - **Web application name:** Name of the Web application diagram.
   - **Display name:** (Optional) Name of the application in the deployment descriptor.
   - **Web application location:** Name of the directory for the diagram.
   - **Components Included in the Web Application:** Existing components that you want to include in the application. Components are organized by four tabs: *Server, JavaServer pages, Taglibs,* and *Web Files.*

Click **Add** to open the file chooser and select JSPs or Web files. Click **Add from project** to open the selection manager and select servlets or TagLibs. To remove an item from the Web application, choose it from the list and click **Remove**.

- **Web application description:** (Optional) Diagram documentation.

5. Click **Finish** to close the dialog and complete the process.

Figure 86 Web application expert



Web application diagram elements

The special Web application design elements are buttons on the diagram toolbar.

Table 42 Design elements on a Web Application diagram

Button	Description
	Servlet Element: Creates a separate deployment properties element. (Use this to represent servlet deployment properties separate from the servlet element or for changing these properties.)

Table 42 Design elements on a Web Application diagram (continued) (continued)

	EJB Shortcuts: Opens a selection manager to add or remove Web shortcuts (to EJBs, servlets, filters, and so on).
	Security Role: A recommended security role for EJB clients; used in the J2EE Deployment Expert. (See Chapter 28, "J2EE Platform Security Support.")
	Principal: A user or a group of users separate from their security role. **Note**: It is possible to change the stereotype using in-place editing.
	Security constraint: Intended protection of the Web contents. Security constraints include Web resource collections, authorization constraints, and user data constraints. (See Chapter 28, "J2EE Platform Security Support.")
	Web resource collection: A set of resources to be protected. A *Web Collection* is a set of URL patterns or HTTP methods that describe resources to be protected.
	EJB Reference: A remote type for EJB reference: EJB or EJB local (for EJB 2.0 specification). (See Chapter 22, "Container Transactions and EJB References".)
	Environment Reference: A static constant, which cannot be changed after deployment.
	Resource Reference: Properties of a referenced resource. Define the type of reference: resource or resource-environment (for EJB 2.0 specification).
	JSP: JSP or servlet.
	JSP Collection: Opens the selection manager to choose multiple JSPs.
	Web Files: Additional Web files such as images.
	Filter mapping: For mapping servlets to filters. Link a servlet to this element and then map it to a filter. (Refer to the Servlets 2.3 specification for details.)
	Error page: A custom error page displayed by the application server when user authentication fails. You can configure an application server to display custom Web pages or other HTTP resources in response to particular HTTP errors or Java exceptions. If the diagram has an Error page element, the deployment descriptor (web.xml) contains a corresponding error-page element, which maps error codes or exception types with the resource path in the Web application.
	TagLib: JSP tag library. (This tag library must already exist.)

Servlet element

A servlet element enables to access the deployment properties of a servlet. You can define servlet deployment properties manually, and link a servlet shortcut with the servlet element.

To create a servlet element on a Web application diagram:

1. Open a Web application diagram and create a shortcut to the servlet whose deployment properties you want to represent.
2. Click **Servlet Element** on the diagram toolbar, then click the diagram.
3. Draw a link from the servlet shortcut to the new servlet element.
4. Open the Inspector of the new servlet element (right click, and choose **Servlet Properties**).
5. Enter the desired values for *Servlet name*, *Load on startup*, *Servlet initialization parameters*, and *Servlet mapping*.

Note The fields in the EJB properties element inspector vary according to the EJB that links to it.

If the EJB properties element links to a CMP entity EJB, the inspector shows database connection properties. You can change the database pool name and the name of the table corresponding to the CMP bean.

The EJB properties element provides the flexibility of using the same EJB in different enterprise or Web applications without modifying the EJB code. It is not necessary to change deployment properties such as JNDI names inside the bean. Instead, you can simply create an additional deployment properties element.

You can use the deployment properties element to refer to this EJB at a later time. For example, you can place a shortcut to the deployment properties element on another EJB assembler or Web application diagram. This effectively includes the corresponding EJB in the other EJB module or Web application.

Note When Together deploys an EJB that is linked to a deployment properties element, the information in the deployment properties element overrides the corresponding information in the EJB.

Web application diagram properties

You can access the properties of a Web application diagram through its Inspector - right click the background of the diagram and choose **Properties.** The Inspector has a special **Web Properties** tab, which is organized into three additional tabs at the bottom of the Inspector: **General, Login config,** and **App event listeners.**

General tab

The **General** tab contains the following fields:

- **Module name:** Name of the `*.war` file created during the deployment process.
- **Welcome File List:** (Optional) `<welcome-file-list>` A list of files to be used for default welcome pages. When the requested URL is a directory, the application server goes through the list, using the first file that it finds for a welcome page.

- **Session time-out:** (Optional) Time in seconds that an application server waits before timing out a session. Minimum value is 1, default is 3600, and maximum value is integer `MAX_VALUE`. This overrides the `<session-descriptor>` element in `web.xml`.
- **Context Root**: (Optional) A string that is used in place of the actual name of the Web application when server resolves a request for the Web application. The server interprets the path name, which must begin with a slash (/), as relative to the context root.
- **Servlet Context Parameters:** (Optional) Defines context initialization parameters of the Web application servlets. The following methods access these parameters:

  ```
  javax.servlet.ServletContext.getInitParameter()
  javax.servlet.ServletContext.getInitParameterNames()
  ```

- **Mime type mapping:** (Required) Defines a mapping between an extension and a mime type. Enter the mime type mappings directly or add them individually using
  the extended editor button .
- **Display name** and **Small/Large icon:** (Optional) For GUI tools, the name and location of small (16x16 pixels) /large (32 X 32 pixels) `*.gif` or `*.jpg` image.

Login Config tab

The **Login Config** tab contains the Web properties relevant to security. It contains the following fields:

- **Authentication method**: (Optional) Authentication method, with choices NONE, BASIC (browser authentication), FORM (html form), and CLIENT-CERT.
- **Realm name:** (Optional) Realm name referenced for user authentication.
- **Form login page:** `<form-login-page>` (Required if authentication method is FORM) Full pathname of a JSP, HTML page, or servlet that generates an HTML form for user certification.
- **Form error page:** `<form-error-page>` (Required if authentication method is FORM) URI of a page to be used as a login error page.
- **Location form login/error page:** (Enabled only if the authentication method is FORM or CLIENT-CERT) Location of the login page (part of the `form-login-page` element of the deployment descriptor).The path begins with a leading slash (/) and is interpreted relative to this root.

For additional instructions on how to provide security, see Chapter 28, "J2EE Platform Security Support."

App event listeners tab

The App event listeners tab contains four listener fields: *Servlet context listener, Servlet context attribute listener, Http session listener,* and *Http session attribute listener.*

For instructions on using the app event listeners tab, see "Associating listeners with Web applications" on page 376.

Creating shortcuts to applets, EJBs, EJB properties, container transactions, and references

Web application diagram cannot contain EJBs. You can, however, place an EJB shortcut on the diagram. During deployment, EJBs are included in the WAR file as ordinary classes.

Note You should use EJB modules (EJB assembler diagrams) to deploy EJBs. Information on EJB modules is in Chapter 21, "Assembling EJB Modules."

If servlets or other elements of your Web application use EJBs, you must place shortcuts to the home and remote interfaces of the EJBs on the Web application diagram. If servlets or other elements of your Web application use EJB properties, container transaction elements, or references, you must place the corresponding design elements (or their shortcuts) on the Web application diagram.

To create a shortcut to an applet, EJB, EJB interface, EJB property, container transaction element, or reference on a Web application diagram:

1. Right-click the background of the Web application diagram and choose **New | Shortcuts**. This opens a selection manager dialog.
2. In the selection manager, choose the elements you want to add from the available content listed on the left and click **Add.** To remove an element from the existing or ready to add elements listed on the right, choose it and click **Remove**.
3. Click **Ok** to close the selection manager and place the shortcuts on the diagram.

For more information on how to work with EJB properties, container transaction elements, and references, see Chapter 22, "Container Transactions and EJB References".

After you populate a Web application diagram with the appropriate components, you can open the J2EE Deployment Expert to generate the deployment descriptor and deploy the application a server. The J2EE Deployment Expert places all of the elements on the Web application diagram into a single WAR file. For more details on deploying and the J2EE Deployment Expert, see Chapter 27, "J2EE Deployment".

Developing applets

Applets are not a part of J2EE technology. However, Web application diagrams may contain shortcuts to applets.

Creating an applet

The first step in creating an applet consists of creating or opening a project for a listener.

To create an applet:

1. Create a project for a listener or use an existing project.
2. To create a new applet, click **Class by Pattern** from the diagram toolbar.
3. In the resulting dialog, choose the *Applet* pattern (*UI Components - Applet*).

   The applet pattern contains the standard set of methods: `init()`, `start()`, `stop()`, `destroy()`, `getAppletInfo()`, `initGUI()`.
4. Enter a name for the applet and click **Finish**.
5. Edit the source code to add the required functionality. Add the code to the skeleton in place of the green comments "`// Write your code here.`"
6. Correct the properties of your applet using the Inspector (choose **Properties** on the applet's right-click menu).

Deploying an applet

To deploy an applet to an application server:

1. Create a Web application diagram or open an existing one.
2. Click the **Add Shortcut** button on the toolbar to open the file chooser dialog.
3. Place the shortcut of the applet on the Web application diagram.
4. Add other classes if necessary.
5. Invoke the J2EE Deployment Expert (see Chapter 27, "J2EE Deployment"). Continue the deployment process as instructed in the Expert.

Working with servlets

A servlet is a Java class on the server side that handles client-server interactions. Usually, these are http-based interactions between Internet browsers and Web servers. Any application capable of creating http-requests can act as a client.

A single servlet can respond to one or multiple types of requests. It is possible to create different configurations: allocate one servlet on the server to handle all incoming requests, or allocate numerous servlets for each request type.

Note When you add a servlet to a project, you may need to add `$TGH$/bundled/tomcat/lib/servlet.jar` to the Searchpath/Classpath in the project properties.

Creating a servlet

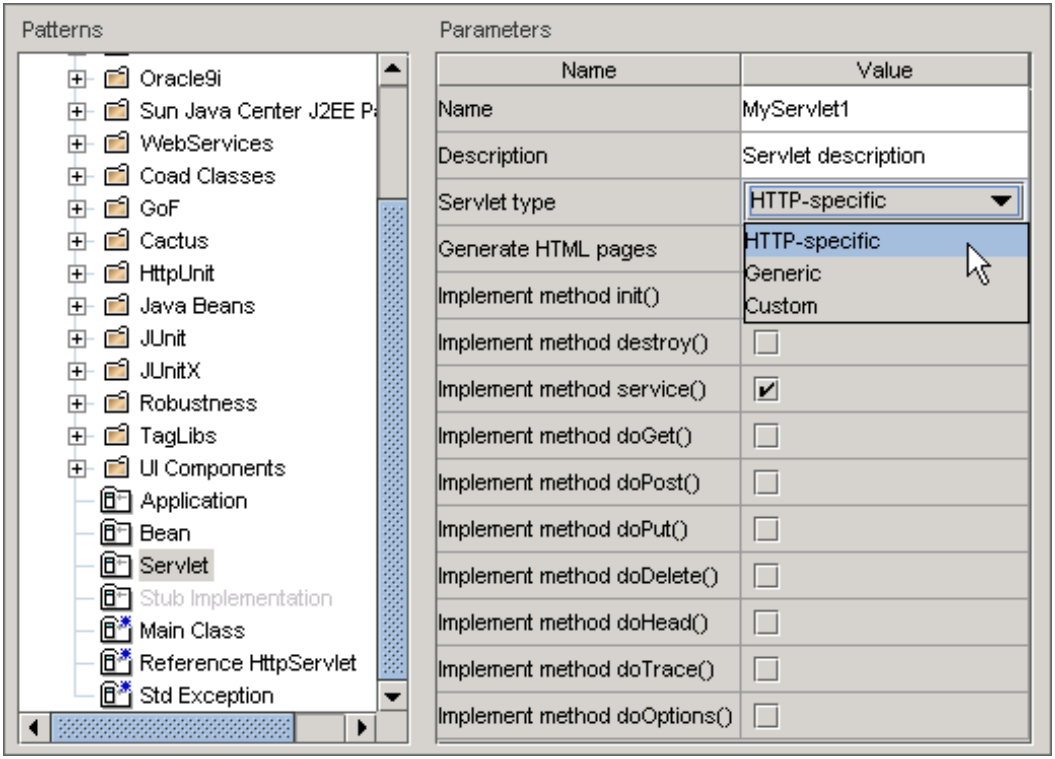
Together offers two convenient ways to create servlets: 1) apply a servlet pattern to a new class or to an existing class, and 2) use the Object Gallery.

Servlets reside on class diagrams. A servlet shortcut can belong to a Web application diagram.

To create a servlet on a class diagram by using a pattern:

1. Open the class diagram where you want to place the servlet.
2. Click the **Class by Pattern** button on the diagram toolbar. This opens the dialog shown in Figure 87.
3. From the list of *Patterns* on the left pane, choose *Servlet*.
4. In the Properties on the right pane, enter the servlet *Name* and *Description.* Choose the *Servlet type* from the dropdown list:
   - **Http-specific servlet:** Extends `HttpServlet`.
   - **Generic servlet:** Extends `GenericServlet` and services any request-response protocol.
   - **Custom Servlet:** Implements the `Servlet` interface.
5. Check the methods that you want generated in the body of the class.
6. If desired, check *Generate HTML pages* to add the code to the request-processing methods (`service, doGet, doPost, doPut, doDelete`) to return an empty HTML page.
7. Click **Ok** to close the dialog and generate the servlet.

Figure 87 Creating a new servlet by pattern



To create a servlet using the Object Gallery

1. Open the class diagram where you want the servlet to belong.
2. On the main menu, choose **File | New.** This opens the Object Gallery.
3. In the Object Gallery, choose *Web* from the Categories on the left pane. Then choose the *Servlet* icon from the Templates on the right pane.

   The resulting *New Servlet* page is an expert that guides you through the process of creating the servlet.
4. Enter the name of the new servlet and its location (the full path).
5. Optionally, to create a shortcut to the servlet on a Web application diagram, check *Create a shortcut to the Servlet class on a Web Application diagram* and enter the name of a new diagram or select an existing Web application diagram.
6. Click **Finish** to create the servlet and the optional shortcut.

The resulting servlet extends `HttpServlet`. The servlet has two methods: `getServletInfo` and `service`.

Putting a servlet into a Web application

The first step in putting a servlet in a Web application is to open the corresponding Web application diagram or create a new one if none exists.

To put a servlet on a Web application diagram:

1. Create a Web application diagram or open an existing one.
2. Click the **Add Shortcut** button on the toolbar.
3. In the left pane, navigate to find the servlet. Choose the servlet and click **Add**, then click **Ok**.

Together includes the servlet among the Web components when it deploys the diagram to an application server.

Working with JSPs

JSP pages provide a presentation layer for Web applications by combining static HTML with dynamically-generated HTML. JSP pages compile into servlets at runtime. A JSP file is not pure Java code; it contains tags and Java scriptlets that can generate an HTML page.

Web application diagrams can contain JSP elements.

To add an existing JSP to the Web application:

1. Click the JSP button on the diagram toolbar or right-click the diagram background and choose **New | JSP**. This creates a JSP element on the diagram.
2. Right-click the JSP in the diagram and choose **Properties** to open its Inspector.
3. In the **JSP Properties** tab of the Inspector, enter the pathname of the existing `*.jsp` file or click the file browser to find it.

When you click on the JSP element in the diagram, The corresponding `*.jsp` file opens in the JBuilder Editor.

To add several JSPs to the Web application diagram at the same time:

1. Click the JSP collection button on the diagram toolbar or right-click the diagram background and choose **New | JSP Collection**.
2. The file chooser dialog immediately opens for you to choose which files you want to add. CTRL+Click to choose each `*.jsp` that you want to include.

The diagram displays a separate node for each JSP chosen.

When you create a JSP diagram element using the Object Gallery, you have the option of creating a new JSP file to correspond to the element on the diagram.

To create JSPs using the Object Gallery:

1. On the main menu, choose **File | New** to open the Object Gallery.

2. Choose *Web* from the categories on the left. Then choose *JSPs* from the templates on the right.
3. Click **Next.** This opens an expert that guides you through creating a JSP.
4. Fill in the fields of the expert.

   Creation Scenario: You can import existing JSPs using the file chooser or you can create a new JSP. If you create a new JSP, enter its name and location.

   Options: Check *Show JSPs on Web Application diagram* to include the JSPs in a Web application. In that case, you can create a new diagram or select an existing one.
5. Click **Finish**. This creates the new Web Application diagram (or opens the existing Web Application diagram) and places the JSP element on it.

Working with tag libraries

JSP pages have tags for interacting with Java objects residing on the server. These include custom tags, which reside in tag libraries, or taglibs.

When a JSP page with custom tags is compiled into a servlet, the custom tags become operations on server-side objects called tag handlers. Tag handlers must either implement `Tag` or `BodyTag` or extend `TagSupport` or `BodyTagSupport`. There is an additional handler that extends `TagExtraInfo` class.

Together has a special taglib diagram for modeling the handlers in a single tag library. The elements on a taglib diagram are shortcuts to the actual handler classes, which reside on class diagrams. Together also has class patterns for tag handlers.

Note When you add a TagLib to a project, you may need to add `$TGH$/bundled/tomcat/lib/servlet.jar` to the project Searchpath/Classpath.

Creating tag handlers on class diagrams

You can create tag handlers on a class diagram by applying a tag handler to an existing class or to a new class.

To create tag handler as a new class on a class diagram:

1. Open the class diagram where you want to place the tag handler.
2. Click the Class by Pattern button on the diagram toolbar.
3. In the resulting dialog box, choose *TagLibs - <handler type>* from the patterns listed on the left.
4. Enter the name of the class in the *Name* field on the right.
5. Click **Finish** to close the dialog and create the tag handler class,

You can place a shortcut to the tag handler on a taglib diagram if desired.

TagLib diagrams

The Together taglib diagram enables the easy creation of handlers and declaration of the tags in a tag library descriptor.

Diagram elements

Table 43 shows taglib diagram elements. When you place an element on a taglib diagram, Together creates a new class and places the shortcut to the class on the taglib diagram.

Table 43 Main elements of taglib diagrams

Icon	Description
	Implements Tag: Handler for a simple tag without a body.
	Extends TagSupport: Handler for tag with attributes.
	Implements BodyTag: Handler for a tag with a body that contains tags, scripting elements, HTML text, and so on between start and end tags
	Extends BodyTagSupport: Handler for a body tag with attributes.
	Extends TagExtraInfo: provides information to the JSP container about the scripting variable.

Taglib diagram properties

The properties of a taglib diagram are defined on its inspector. You can open the inspector by right-clicking the background of the diagram and selecting **Properties.**

In addition to properties common to all diagrams, the taglib diagram inspector has a **TagLib properties** tab that defines the tag library. This tag includes the following properties:

- **TLD File Name:** Fully qualified name of the tag library descriptor that is generated by the **Generate TLD** command of the diagram right-click menu. You can enter the information in this field directly or select it using the file chooser button.
- **Short name:** A logical name that indirectly references the TLD (mapped to an absolute address in the deployment descriptor).
- **TagLib Version:** Version of the tag library.
- **JSP Version:** JSP specification version used by the tag library.

- **TagLib URI:** URI that uniquely identifies the tag library.
- **TagLib location:** Physical location of the tag library.
- **Use existing TLD:** Checking disables the **Generate TLD** command of the diagram right-click menu
- **Display name/Small icon/Large icon:** Equivalent to the same properties for Web application diagrams. (See "Web application diagram properties" on page 363.)

Properties of the handlers

You can open the Inspector of a tag handler class by right clicking the handler element on the class diagram or taglib diagram and choosing **Properties.**
The **Tag properties** tab of the Inspector contains the tag details of the handler. These details include the following:

- **Tag name:** The name of the tag as known by the JSP.
- **Body content:** Possible values are *JSP, EMPTY,* and *TAGDEPENDENT.*
- **TagExtraInfo class:** A separate class containing additional information associated with the tag handler.

Creating and using tag libraries

The three basic steps of creating and using tag libraries are as follows:

Step 1: Create and populate the taglib diagram.

Step 2: Generate the tag library descriptor.

Step 3: Add the tag library to the Web application diagram.

To create and populate a taglib diagram:

1. Create the taglib diagram using either the **New Diagram** button on the Designer toolbar or the Object Gallery. *TagLib* is a diagram type listed on the Together tab.
2. Populate the taglib diagram with the necessary tag handlers (shortcuts).
3. Open the inspector for each diagram element:
   - Fill in the inspector fields on the **Tag properties** tab: tag name, implemented `TagExtraInfo` class, and tag body content (if any).
   - Enter additional information as needed in the **Description** tab.

To generate the tag library descriptor:

1. Right-click on the diagram background to open the taglib diagram Inspector.
2. Click the **TagLib properties** tab on the top then the **General** tab on the bottom of the Inspector.

3. In the *TLD file name* field, enter the path name of the tag library descriptor, or locate it using the file chooser. Fill in the other fields as required.
4. Right-click on the diagram background and choose **Generate TLD.**

To add the tag handlers to the Web application diagram:

1. Place a **TagLib** element on the Web application diagram.
2. In the resulting dialog, expand the nodes on the left pane and choose the tag library to be added.
3. Click **Ok** to close the dialog and place a shortcut to the tag library on the Web application diagram.

Working with filters

Filters are servlets in Web applications that can modify requests or responses. Filter mappings determine how filters apply to servlets and static Web resources.

Together supports filters and filter mappings through the filter mapping elements on Web application diagrams.

Creating filters and filter mappings

Filters are classes that reside on class diagrams. Web application diagrams can contain shortcuts to filters.

To create a filter or a collection of filters in a Web application diagram:

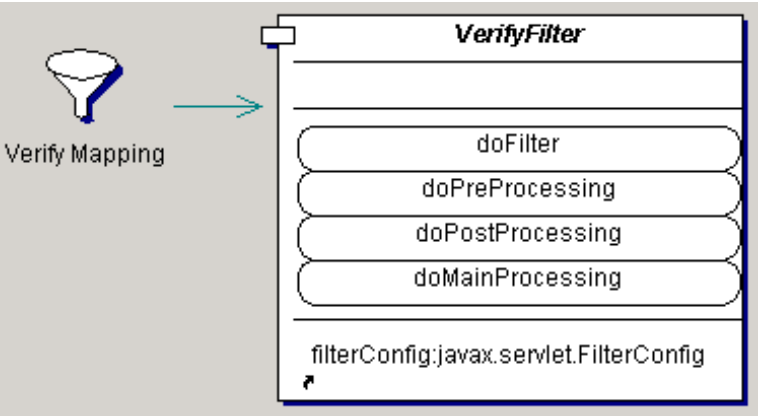
1. Place a **Filter Mapping** element on the diagram.
2. Open the filter mapping Inspector: right-click the filter mapping element and choose **Properties**.
3. In the Inspector, choose the **Mapped Filters** tab.
4. Click **Add** to create a filter. You can edit its name in the list of filters on the tab.

This process generates a filter class on the default diagram that implements the `javax.servlet.Filter` interface. Each filter can process the current request, the current response, and the `FilterChain` of filters to be executed.

The Web application diagram displays a filter mapping, a shortcut to the filter class, and a Web link connecting the two, as illustrated in Figure 88.

Note When you add filters to a project, you may need to add `$TGH$/bundled/j2ee/lib/j2ee.jar` to the Searchpath/Classpath in the project properties. Filter mapping is defined only for application servers supporting EJB 2.0 specification.

Figure 88 Filter mapping on a Web application diagram



Setting filter and filter mapping properties

You can access the inspector of a filter class through its node on a class diagram or through its shortcut on the Web application diagram. Right-click the filter on the diagram and choose **Properties.**

The filter class inspector has a **Filter Properties** tab, where you can specify the following properties:

- **Filter name:** Maps the filter to a servlet or URL.
- **Display name:** Visible name of the filter.
- **Icon:** Fully-qualified path to an image file used as the filter's icon.
- **Initialization parameters:** Filter initialization parameters. Click the chooser button at the right of the field to open a dialog box for adding or removing parameters.

The filter mapping inspector has two special tabs: **Mapped filters** and **URL Pattern Mapping.**

- **Mapped filters:** For adding filters (as described in the previous section) or removing links. The **Remove** button deletes the link between a filter mapping and a filter. It does not delete the filter class and it does not delete the shortcut to the filter class on the Web application diagram.
- **URL Pattern Mapping:** Defines groups of servlets and static resources in the Web application to which the filter is applied. The rules of the path mapping are outlined in the Java Servlet Specification version 2.3, SRV 11.1, *Use of URL Paths*.

Deploying filters

During deployment, Together archives filters to a `*.war` file and deploys to the application server. For each filter element Together generates the corresponding lines in the deployment descriptor.

Working with listeners

You can monitor a servlet by creating a listener to respond to servlet events. The listener is a class that implements a servlet event listener interface. Servlet event listeners can notify for state changes in `ServletContext` and `HttpSession` objects.

Note Listeners are defined only for application servers supporting the EJB 2.0 specification.

Supported event types

A listener can notify the rest of a system when an event takes place (for example, the Web server shuts down or the application is removed from the Web server).

Together supports the following event types:

- **Servlet Context Events**
  - **Lifecycle:** The servlet context has just been created and is available to service its first request or the servlet context is ready to be shut down.
    `javax.servlet.ServletContextListener` has these methods:
    - `contextDestroyed(ServletContextEvent)`
    - `contextInitilized(ServletContextEvent)`
  - **Changes to attributes**: Attributes on the servlet context have been added, removed, or replaced.
    `javax.servlet.ServletContextattributesListener` has these methods:
    - `attributeAdded(ServletContextAttributeEvent)`
    - `attributeRemoved(ServletContextAttributeEvent)`
    - `attributeReplaced(ServletContextAttributeEvent)`
- **HttpSession Events**
  - **Lifecycle:** An HttpSession has been created, invalidated, or timed out
    `javax.servlet.http.HttpSessionListener` has these methods:
    - `sessionCreated(HttpSessionEvent)`
    - `sessionDestroyed(HttpSessionEvent)`
  - **Changes to attributes:** Attributes have been added, removed, or replaced on an HttpSession.
    `javax.servlet.http.HttpSessionattributesListener` has these methods:

- attributeAdded(HttpSessionBindingEvent)
- attributeRemoved(HttpSessionBindingEvent)
- attributeReplaced(HttpSessionBindingEvent)

Creating a listener

The first step in creating a listener is to determine the class diagram (or the package) for the listener.

To create a listener:

1. Open the class diagram for the listener. Then click **Class by Pattern** on the toolbar and click the diagram. This opens the Choose Pattern dialog.
2. In the patterns displayed on the left, expand *J2EE - App Event Listeners.*
3. Choose one of the four listener patterns and enter a name on the right.
4. Click **Finish.**

Associating listeners with Web applications

You can include listeners as properties of Web application diagrams.

To specify a listener as a Web application property:

1. Create a Web application diagram or open an existing one. Then open its inspector by right-clicking the diagram and choosing **Properties.**
2. Choose the **Web Properties** tab on the top of the inspector and then choose the **App event listeners** tab on the bottom. The **App event listeners** tab has four fields, one for each type of listener.
3. At the right of the field for the appropriate listener type, click the button to open a multi-string editor. The resulting dialog displays the listeners of that type.
   - Click **Add** to add a listener. This opens a new line in the listener display. Clicking the button at the right side of a line opens a *Select Element* dialog for choosing the listener. You can add multiple listeners by repeating this process.
   - To remove a listener from the list, choose the listener and click **Remove**.
   - The order that the listeners are listed is the order in which they are processed. To change the processing order, choose a listener and click **Up** or **Down** as needed.
4. Click **Ok** to close the dialog.

The **App event listeners** tab displays the listeners for each type, separating listeners of the same type by commas.

Deploying a Web application

After you populate a Web application diagram with the desired components, you can set its deployment properties in the *Web Properties* tab of the diagram Inspector.

Note If the target application server for your project is Borland Enterprise Server 5.2, the inspector for a Web application displays the **Borland Enterprise Server 5.2** tab. Use this tab for setting additional deployment properties. For more detail information on the contents of the Borland Enterprise Server 5.2 tags, see **http://info.borland.com/techpubs/books/bes/pdfs52/.**

open the J2EE Deployment Expert by choosing **Deploy | J2EE Deployment Expert** from the main menu. The J2EE Deployment Expert generates a deployment descriptor, creates a WAR file from all of the elements on the Web application diagram, and deploys the application to a server.

For more details on how to use the J2EE Deployment Expert, see Chapter 27, "J2EE Deployment".

Note You can archive several Web applications and EJB modules in one EAR and deploy it to the current application server. For information, see Chapter 24, "Designing and Developing Enterprise Applications".

For "fast track" deployment with default values or deployment prototyping, you can use the J2EE Deployment Expert from an appropriate class diagram. For "full-featured assembly information" with control over security and permissions, you should use the J2EE Deployment Expert from the Web application diagram.

Chapter

24

Designing and Developing Enterprise Applications

Together provides a special enterprise application diagram to visually assemble enterprise applications for deployment. This chapter discusses designing and developing enterprise applications by using diagrams. The chapter includes the following topics:

- "Visual assembly of enterprise applications for deployment" on page 379
- "Deploying an enterprise application" on page 384

Visual assembly of enterprise applications for deployment

Together provides a special *Enterprise application* diagram for collecting elements for an EAR file into a single diagram. The enterprise application diagram combines Web applications, EJB modules, classes, and connectors. You can use it to visually assemble an enterprise application for deployment.

An enterprise application diagram can contain shortcuts to EJB assembler, Web application, resource adapter, and class diagrams. You can include existing WAR, JAR, and RAR archive files on the diagram as well. An enterprise application diagram provides all necessary elements for generating the deployment descriptor and creating an EAR archive file.

Note If the target application server does not support `*.ear` generation (such as BEA WebLogic 5.1 and IBM WebSphere 3.5), multiple `*.jar` files are generated instead. Otherwise, a single `*.ear` file is generated when deployment is performed against the enterprise application diagram.

Enterprise application diagrams also support security roles. (See Chapter 28, "J2EE Platform Security Support" for more information.) Thus, an enterprise Application diagram is a visual equivalent of the enterprise application described by the EJB 2.0 Specification. The diagram provides application assembly information to the application deployer, which the J2EE Deployment Expert in Together.

Creating enterprise application diagrams

There are two ways to create an enterprise application diagram:

- Create a new enterprise application diagram using the Object Gallery or the diagram toolbar button and then add enterprise components to it.
- Create a complete enterprise application using the Object Gallery. If you already have existing components for a single enterprise application, you can use this to create an enterprise application diagram that is ready for deployment.

To create a new enterprise application diagram:

1. Choose **File | New** on the main menu.
2. In the Object Gallery, choose *General* in the categories on the left, and choose the *Diagram* icon in the templates on the right. Click **Next** to select the type of diagram.
3. Open the Together tab, choose the *Enterprise Application* icon, and click **Finish**.

Alternatively, you can click **New Diagram** on the diagram toolbar, go to the Together tab, and select **Enterprise Application.**

At this point, you can develop the content of the diagram using visual design elements from the toolbar.

You can use the Object Gallery to create a new enterprise application for existing components if you have several EJB modules and Web applications that you want to combine in the same module for deploying.

To create an enterprise application with the Object Gallery:

1. Choose **File | New** on the main menu.
2. In the Object Gallery, choose *Enterprise* in the categories on the left.
3. Choose the *Enterprise Application* icon in the templates on the right.
4. Click **Next.**

The resulting dialog box is an expert that guides you through the process of creating the enterprise application. Follow its recommendations, filling in the most important parameters, inserting names, and choosing paths as needed. You do not need to work extensively with the enterprise application diagram - it is automatically created and populated with shortcuts to the components that you specify.

Enterprise application diagram elements

All enterprise application diagram elements are listed in the toolbar on the left side of the diagram. Table 44 lists the special elements for enterprise application diagrams:

Table 44 Enterprise application diagram design elements

Icon	Description
	Module: A shortcut to an existing EJB assembler, Web application, resource adapter, application client, or class diagram.
	Archived Module: A shortcut to an existing archive file.
	Security Role: The *security role* that stands for one of the recommended security roles for the application clients. The *Security Role* element presents a simplified view of enterprise applications security to the application deployer (such as the J2EE Deployment Expert).

Properties of an Enterprise Application diagram

You can set or change properties of an enterprise application diagram using the diagram's Inspector.

To invoke the Inspector and to set the diagram properties:

1. Right-click on the diagram background and choose **Properties**.
2. Click the **Properties** tab to set the diagram name.
3. Click the **Enterprise Properties** tab to set specific enterprise properties used during the deployment process:
   - **Module name:** Name of an archive `*.ear` file that is created during the deployment process. By default, this name is the same as the *Name* on the Properties page.
   - **Display name, Small icon, Large icon:** Optional parameters for the deployment descriptor.

Creating diagram shortcuts

You can assemble one or more of your project's existing Web application, EJB assembler, and resource adapter diagrams into an enterprise application diagram.These other diagrams are added to the enterprise application diagram as shortcuts.

To create a shortcut to a diagram:

1. Create a new enterprise application diagram or open an existing one.

2. To include diagrams that are not part of the current project, specify their paths in the *Search/Classpath* tab of the Project Properties dialog (**Project | Project Properties**).

3. Click **Module** on the diagram toolbar and place the element on the designer pane. (Alternatively, right click the diagram background and choose **New | Module**.)

   This opens a section manager dialog where you can choose available EJB assembler, Web application, resource adapter, or class diagrams to be added to the enterprise application diagram.

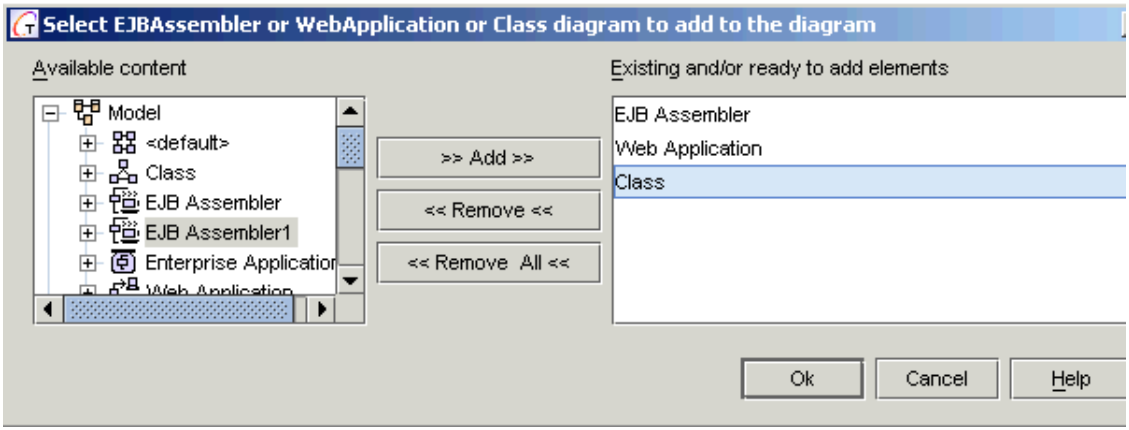
4. In the *Add Module* dialog, expand the *Model* node and locate the EJB classes and interfaces from your project that you want to display in the diagram. Select them in the tree view and click the **Add** button.

5. Also in the *Add Module* dialog, expand the *Search/Classpath* node and locate the EJB modules, Web applications, or classes from outside your project (if any) to display in the diagram as shown in Figure 89. Select them in the tree view and click the **Add** button.

Note Even though any shortcut may be physically present on an enterprise application diagram, you should place only "allowed" shortcuts on an enterprise application diagram. Otherwise, the deployment will fail.

6. To include diagrams that are not part of the current project, specify their paths in the *Search/Classpath* tab of the Project Properties dialog (**Project | Project Properties**).

Figure 89 Adding elements to an enterprise application diagram



Importing existing archive files

You can import existing JAR, WAR, and RAR archive files to an enterprise application diagram.

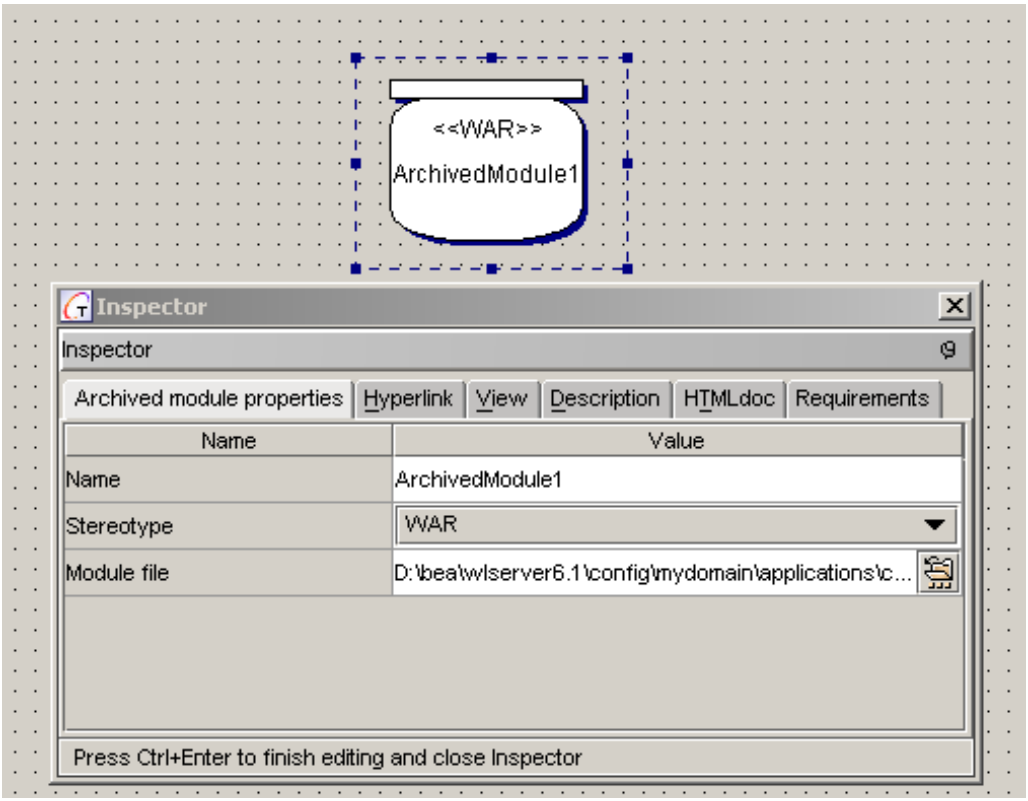
To add existing JAR, WAR or RAR archive modules to an enterprise application diagram:

1. Click **Archived Module** and place this element on the diagram.
2. Open the Inspector to define the name of this element and its other properties. Choose *Stereotype* (JAR,WAR or RAR), and use the file chooser to find the full path to the corresponding WAR, JAR, or RAR file.

Figure 90 shows a WAR file Inspector along with the WAR element on the diagram.

Note Import of the archive files is possible only for the standard deployment descriptors. Vendor-specific descriptors (such as `weblogic-ejb-jar.xml`) are ignored; the information contained within them is not used. You can import only Generic 1.1 and Generic 2.0 archive files without restrictions.

Figure 90 Inspector of a WAR file.



Deploying an enterprise application

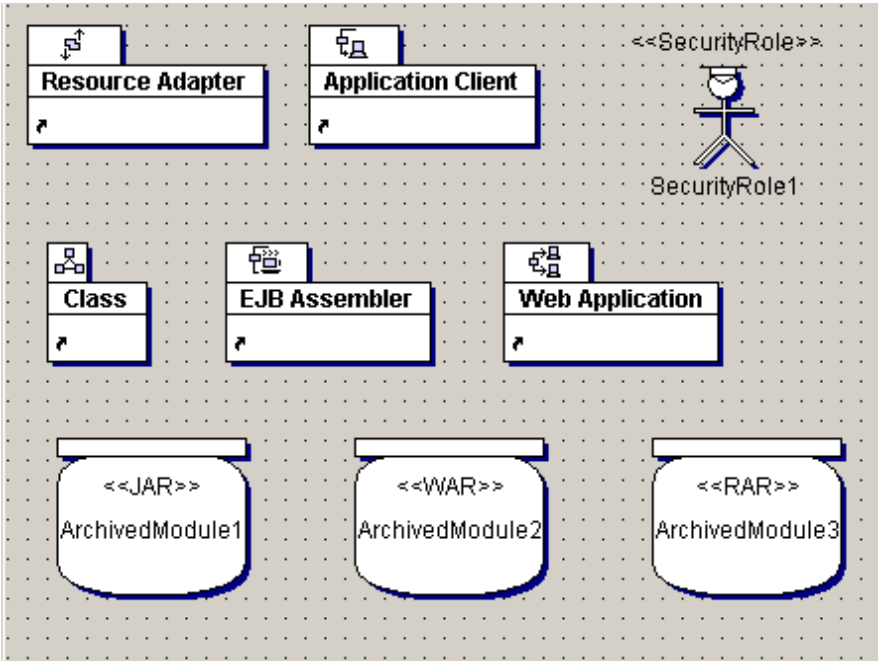
Together's deployment expert simplifies the process of deploying EJBs to various application servers.

To deploy an enterprise application:

1. Place all necessary shortcuts to EJB assembler, Web application, resource adapter, application client, and class diagrams and other design elements on the enterprise application diagram.
2. Define a security role for this diagram if desired.
3. From the main menu, choose **Deploy | J2EE Deployment Expert**. Then follow the steps outlined by the J2EE Deployment Expert.

   Figure 91 shows an enterprise application diagram that has been populated with all possible elements.

Figure 91 Enterprise application diagram



4. After you populate an Enterprise application diagram with the desired shortcuts, you can set its deployment properties in the *Enterprise Properties* tab of the diagram Inspector.

Note If the target application server for your project is Borland Enterprise Server 5.2, the inspector for an Enterprise application displays the Borland Enterprise Server 5.2 tab. Use this tab for setting additional deployment properties. For more detail information on the contents of the Borland Enterprise Server 5.2 tags, see **http://info.borland.com/techpubs/books/bes/pdfs52/.**

5. From the main menu, choose **Deploy | J2EE Deployment Expert**. Then follow the steps outlined by the J2EE Deployment Expert.

Chapter

25

Designing and Developing Application Clients

Together provides special application client diagrams for modeling clients of enterprise applications. This chapter discusses how to use application client diagrams in preparation for deploying enterprise applications.

This chapter includes the following topics:

- "Modeling enterprise application clients" on page 387
- "Using an application client diagram" on page 391

Modeling enterprise application clients

Together provides a special *application client* diagram for modeling elements of an enterprise application client. Elements on an application client diagram include shortcuts to the classes and the references to EJBs and external resources required to deploy the client.

Application clients are packaged in JAR format files with the `*.jar` extension. They include a deployment descriptor describing the enterprise beans and external resources that the application references. Access to resources is configured at deployment time.

An application client diagram contains all the necessary elements needed for deployment. The Together J2EE Deployment Expert can generate the deployment descriptor, adding it and the client to the resulting JAR archive.

Creating application client diagrams

If your project has application client classes, you can create an application client diagram and place shortcuts to classes on this diagram. You can also create references to additional files, references to EJBs, and so on.

There are two ways to create an application client diagram:

- Create a new application client diagram using the Object Gallery or the diagram toolbar button and then add components to it.
- Create an application client using the Object Gallery. If you already have existing components you want to combine in a single enterprise application, this creates an enterprise application diagram that is ready for deployment.

To create a new application client diagram:

1. Choose **File** | **New** on the main menu.
2. In the Object Gallery, select **General** in the categories on the left, and select the **Diagram** icon in the templates on the right. Click **Next** to select the type of diagram.
3. Click the Together tab, click the **Application Client** icon, and click **Finish**.

Alternatively, you can click the **New Diagram** icon on the diagram toolbar, go to the Together tab, and select **Application Client**.

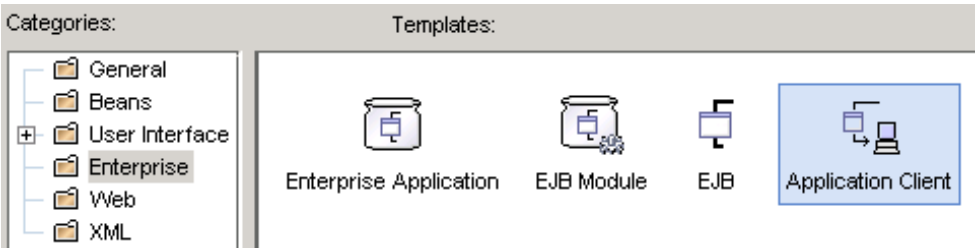
The new application client diagram with the default name *Application Client* opens in the Designer pane. To change the default name, use the inspector (right-click menu | **Properties**). At this point, you can develop the content of the diagram using elements from the diagram toolbar.

You can use the Object Gallery to create a new enterprise application for existing components if you have several EJB modules and Web applications that you want to combine in the same module for deploying.

To create an application client with the Object Gallery:

1. Choose **File** | **New** on the main menu.
2. In the Object Gallery, choose *Enterprise* in the categories on the left. Then choose *Application Client* among the icons on the right, as shown in Figure 92.

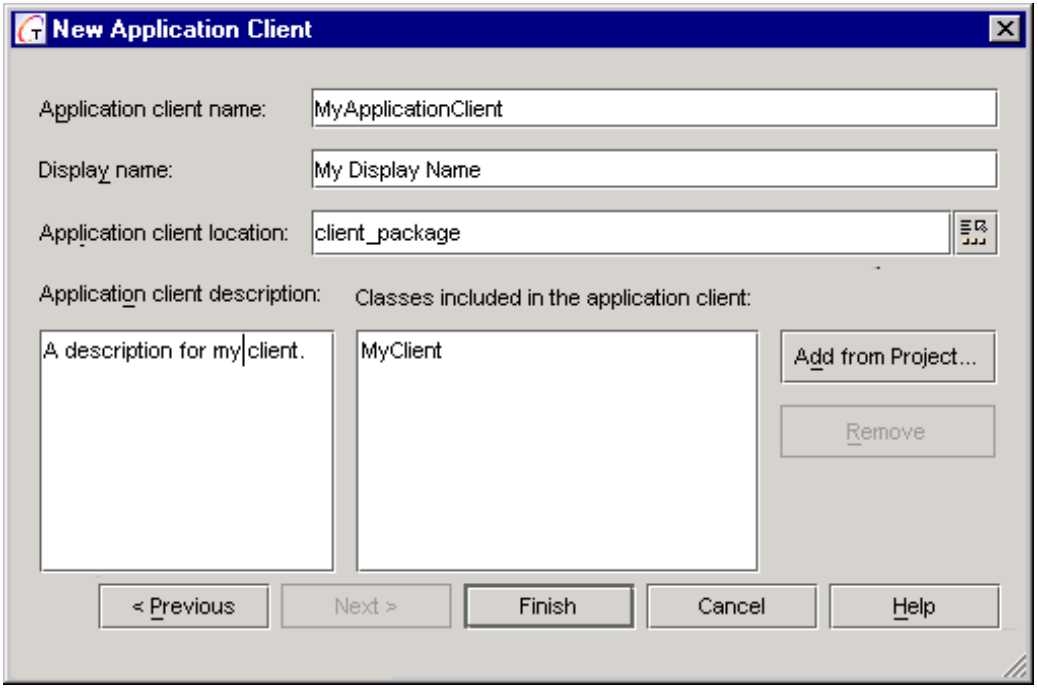
Figure 92 Choosing Application Client in the Object Gallery



3. Click **Next.**

   The resulting dialog box is an expert that guides you through the process of creating the application client. Figure 93 shows the application client expert.

Figure 93 Application client expert from the Object Gallery



   Follow the recommendations in the Object Gallery, filling in the parameters as required:

   - **Application client name:** Name of the application and the new application diagram
   - **Display name:** Display name for the deployment descriptor
   - **Application client location:** Location of the diagram. Enter the full path name or click the file chooser button on the right to select the folder.
   - **Application client description:** Diagram description (on the Description tab of the diagram inspector).
   - **Classes included in the application client:** Click **Add from Project** to open a selection manager dialog for choosing the client classes. To remove class from the list, click the class to select it then click **Remove.**

4. Click **Finish**. This creates the application client diagram that represents your application client.

Visual design elements

Buttons for all application client diagram elements are in the diagram toolbar. The design elements are listed in Table 45. Notice that the elements are either shortcuts or references. An application client diagram cannot contain classes.

Table 45 Application client diagram design elements

Icon	Description
	Client Classes shortcuts: Opens a selection manager dialog box, where you add or remove shortcuts of client classes.
	EJB Reference: Creates a reference to an EJB. Define a remote type for EJB reference: EJB, or EJB local (for the EJB 2.0 specification).
	Environment Reference: Creates an element with the properties of some static constant, which cannot be changed after EJB deployment.
	Resource Reference: Creates an element representing the properties of the referenced resource. Define the type of reference: resource or resource-environment (for the EJB 2.0 specification).

Properties of an application client diagram

You can specify the properties of an application client diagram using the diagram inspector.

To open the inspector and set the diagram properties:

1. Right-click on the diagram background and choose **Properties** on the right-click menu.
2. In the inspector, open the **Properties** tab and set the diagram name
3. Open the **Application Client Properties** tab to set specific EJB properties used during the deployment process.
   - **Module name:** Name of an archive `*.jar` file that is created during the deployment process and then included in an archive `*.ear` file. By default, this name is the same as the *Name* on the Properties page.
   - **Application client main class:** Enter the main class, or browse for it using the file chooser button for selecting the full path to the existing main class.
   - **Display name, Small icon, Large icon:** Optional parameters for the deployment descriptor.

Creating “one-click” application clients

To create a “one-click” application client:

1. Create or open a Together project, and create or navigate to the package where you want to create an application client.

2. Create or open a class diagram in the target package.
3. Create client classes on the class diagram.
4. Create an application client diagram, and name it.
5. Click the **Client Classes Shortcuts** button and place it on the application client diagram. Use the chooser to select the classes to add to the diagram as shortcuts.
6. Add other necessary design elements (such as environment and resource references) on the diagram.

Using an application client diagram

You can use all of the elements placed on an application client diagram as if they constitute a separate module with the same name as the diagram. You can:

- Compile classes from each client module separately.
- Generate a client XML deployment descriptor for each client module.
- Package compiled classes and a deployment descriptor into a JAR for each client module.

Note Generating the deployment descriptor and packaging the application client and its deployment descriptor into a JAR file are possible only from an enterprise application diagram or an EJB assembler diagram.

To start the deployment process:

1. Create or open an enterprise application or EJB assembler diagram.
2. Click **Shortcuts** then the diagram. In the resulting dialog, choose the application client diagram.
3. Place other necessary design elements on the enterprise application or EJB assembler diagram.
4. Choose **Deploy | J2EE Deployment Expert** on the main menu to invoke the J2EE Deployment Expert and start the deployment process (see "J2EE Deployment" on page 399).

Chapter

26

Designing and Developing Resource Adapters

Resource adapters enable Java applications to connect to enterprise information systems. This chapter discusses the Together resource adapter diagrams that support resource adapters and the J2EE connector architecture.

This chapter includes the following topics:

- "J2EE Connector Architecture" on page 393
- "Creating a resource adapter diagram" on page 394
- "Using a resource adapter diagram" on page 397

J2EE Connector Architecture

J2EE 1.3 includes a connector architecture (J2EE CA) for integration of enterprise information systems (EIS) with application servers and enterprise applications. The focus of J2EE CA is integration of J2EE applications with EISs that are not relational databases. JDBC provides the database connectivity.

A *resource adapter* is software that enables a Java application to connect to an EIS. Both JDBC and J2EE CA are examples of resource adapters. Resource adapters collaborate with application servers to provide the connection pooling, transactions, and security between EIS and enterprise applications.

EJBs get resource references of a specific provider through JNDI. They find the necessary resources using `InitialContext.lookup()` and `<res-ref-name>`. This enables the container to control the resources of EJBs.

Together supports resource adapters and the J2EE container architecture through special *resource adapter diagrams*. During deployment Together creates RAR files from resource adapter diagrams.

Important You cannot include a RAR file into an enterprise application that is targeted to the Sun EE 1.3.1 reference implementation.

Creating a resource adapter diagram

If your project already has appropriate `ConnectionFactory` and `ManagedConnectionFactory` classes, you can create a resource adapter diagram.

To create a new Resource Adapter diagram:

1. Choose **File | New** on the main menu.
2. In the Object Gallery, choose *General* in the categories on the left, and click the **Diagram** icon in the templates on the right. Click **Next** to select the type of diagram.
3. Open the Together tab, then click the **Resource Adapter** icon.
4. Click **Finish** to complete the process and create the diagram.

The default name of the new diagram is Resource Adapter. To change the default name use the diagram inspector (right-click menu | **Properties**).

Resource adapter diagram elements

The special resource adapter diagram elements are on the toolbar on the left side of the diagram. Table 46 lists these elements.

Table 46 Resource adapter diagram elements

Icon	Description
	Config Property: Declaration of a single configuration property for a `ManagedConnectionFactory` instance (an option description, a name, a type and an optional value for a configuration property). This element corresponds to the `<config-property>` attribute in the deployment descriptor.
	Authentication Mechanism: A type of an authentication mechanism. Any additional security mechanisms are outside of the scope of the connector architecture. This element corresponds to the `<authentication-mechanism>` attribute in the deployment descriptor.
	Security Permission: Security permission based on the Security policy file syntax. This defines what permissions (which types of system resource accesses) are allowed by code from specified code sources. Refer to the corresponding URL of the security specification. This element corresponds to the `<security-permission>` attribute in the deployment descriptor.

Table 46 Resource adapter diagram elements

Icon	Description
	Connection: Links to a connection interface and to a connection implementation class.
	Connection Factory: Links to a connection factory interface and to a connection factory implementation class.
	Resource Adapter Link: Links Connection/Connection Factory elements with the corresponding interfaces and implementation classes.

A resource adapter diagram contains only shortcuts to connection implementation classes and their interfaces. It does not contain the actual connection classes.

To add a shortcut of an existing connection class:

1. Right click the diagram background and choose **New** | **Shortcuts**.
2. In the resulting selection manager dialog, choose the connection class.

Creating a resource adapter diagram using a pattern

Together provides a special pattern for resource adapters and their associated connection classes.

To create a resource adapter diagram using a pattern:

1. Open the class (or package) diagram where you want to place the connection classes corresponding to the new resource adapter diagram.
2. Click **Class by Pattern** and click the diagram.
3. In the resulting dialog box, choose *J2EE - Resource Adapter* from the Patterns listed on the left.
4. In the Parameters on the right, enter the name for the new resource adapter diagram. For the remaining values, enter the names of new connection classes. If you want to use an existing connection class instead of creating a new one, click the file browser button at the right of the name field.

   Figure 94 shows the parameters for the resource adapter pattern. The new resource adapter diagram will have the default name, Resource Adapter.

Figure 94 Resource adapter pattern parameters

Parameters

Name	Value
Resource Adapter Diagram name	Resource Adapter
Managed Connection Factory class name	myManagedConnectionFactory
Connection Factory interface name	myConectionFactoryInterface
Connection Factory class name	myConnectionFactory
Connection interface name	myConnectionInterface
Connection class name	myConnectionClass

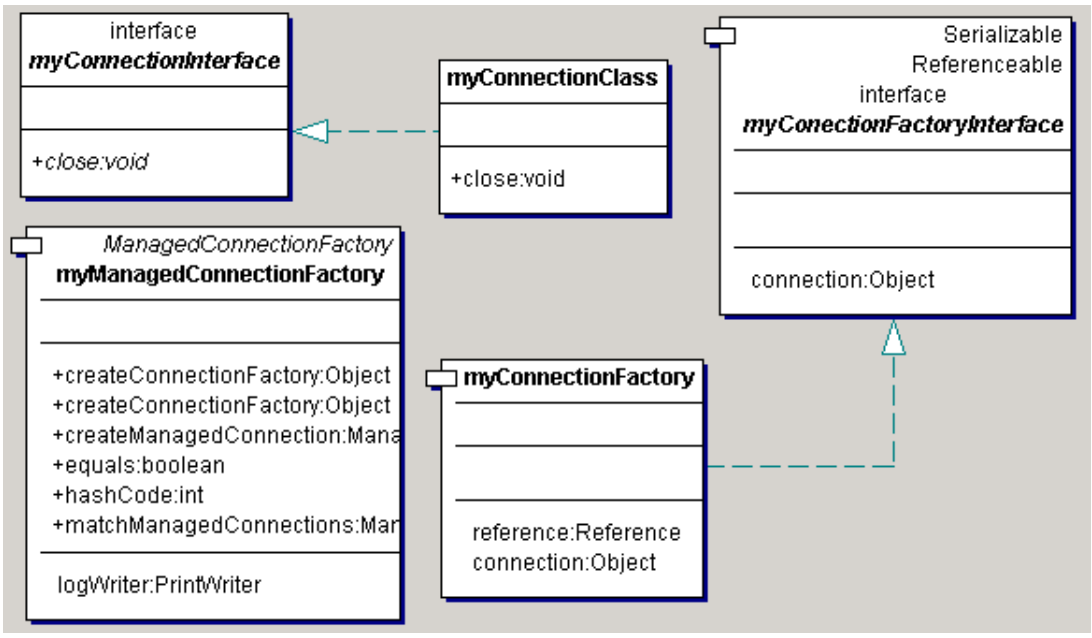
5. Click **Finish**.

The class diagram displays the new connection classes and interfaces. Figure 95 shows the classes created as a result of filling in the dialog box of Figure 94.

The pattern also creates the new resource adapter diagram, as shown in Figure 96. The diagram is part of the class diagram package. Notice that the classes on the resource adapter diagram display with the shortcut symbol in the lower left corner.

You can open the resource adapter diagram from the Explorer. Look for the diagram name and the Resource Adapter icon, .

Figure 95 Class diagram with connection classes



Using a resource adapter diagram

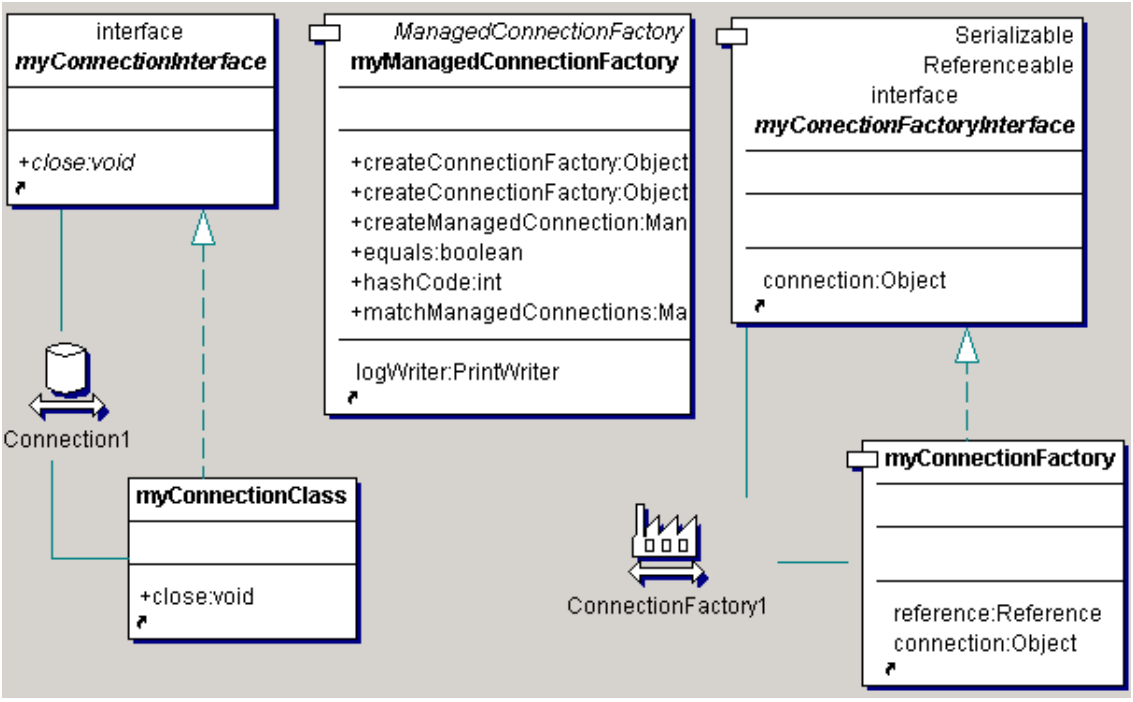
You can consider the elements that are on a resource adapter diagram as a separate module with the diagram's name.

Together provides:

- Generating an XML deployment descriptor for each resource adapter diagram.
- Packaging the compiled classes and a deployment descriptor into an RAR for each resource adapter diagram.
- Deploying to the current application servers, which support J2EE specification, version 1.3.

The deployment process in Together completes all of these tasks.

Figure 96 Resource adapter diagram.



After you populate a Resource adapter diagram with the desired components, you can set its deployment properties in the *Resource Adapter* tab of the diagram Inspector.

Note If the target application server for your project is Borland Enterprise Server 5.2, the Inspector for a Resource adapter displays the **Borland Enterprise Server 5.2** tab. Use this tab for setting additional deployment properties. For more detail information on the contents of the Borland Enterprise Server 5.2 tags, see **http://info.borland.com/techpubs/books/bes/pdfs52/.**

To deploy from a resource adapter diagram:

1. Open the resource adapter diagram.
2. On the main menu, choose **Deploy | J2EE Deployment Expert**.
3. Continue the deployment process, filling in the process options and fields as required by the J2EE Deployment Expert.

For detailed instructions on deployment, see Chapter 27, "J2EE Deployment".

Chapter

27

J2EE Deployment

J2EE deployment is a process of generating deployment descriptors and packing archive files (`*.jar`, `*.war`, or `*.ear`) on an application server. This chapter discusses using the J2EE Deployment Expert to assist in the deployment process.

This chapter discusses the following topics:

- "Deployment process" on page 399
- "Supported application servers" on page 400
- "Installations and resources required for deployment" on page 401
- "Together provides the following plug-ins for starting application servers:" on page 402
- "J2EE Deployment Expert features" on page 404
- "J2EE Deployment Expert options common to all servers" on page 409
- "Server-specific J2EE Deployment Expert pages" on page 418
- "Making transitions among application servers" on page 426
- "Encoding in deployment descriptors" on page 430

Deployment process

Without the aid of an "expert," the process of deploying to application servers can be tedious and difficult. For example, an e-commerce or other enterprise application based on J2EE technology may contain multiple EJBs, JSP files, servlets, and other components. Before the application is ready to use, you must select an application server, write the deployment descriptors (and possibly other necessary files depending on the server), pack all files in the archive, and deploy the result to a specific location.

Together automates this process. By using the J2EE Deployment Expert, you can verify and correct your code according to the EJB specification of the application server, generate deployment descriptors, compile all files, pack them in `*.jar`, `*.war`,`*.rar`, or `*.ear` files, and deploy to the application server. You can optionally generate a simple JSP client for testing the deployed EJB running on the application server.

Together can deploy to most of the popular application servers. On server platforms that support it, Together can hot deploy directly to the application server. You can also start some application servers from within Together.

Supported application servers

When you open a new Java project and then invoke the J2EE Deployment Expert, the dialog opens with default Generic 2.0 settings.

To see a list of the currently supported servers and to select a supported server:

1. Open a project.
2. From the main menu, choose **Project | Project Properties.**
3. In the resulting project properties dialog box, click the **EJB** tab.
4. The drop-down list of application servers shows the server platforms that are currently integrated into Together. Choose your target application server from the list.

Integrating other application servers

It is possible to integrate other application servers and add them to the list of the supported application servers.

To find out information about the application servers that currently can be integrated:

1. Choose **Help | On the Web | Integration Updates** from the main menu.
2. Choose the *J2EE Application Server* hyperlink on the site **http://www.togethersoft.com/developers/integrations/** to see the current list of J2EE Application servers supported in Together.

   This list provides the following information:

   - The level of integration (*High, Medium, Low*).

     If the level of integration for your application server is high, you can deploy directly from within Together using the J2EE Deployment Expert. If the level is not high, download this application server and read the help on the TogetherSoft Web site about the possibilities of integration.

   - Where you can find this application server (*In Build, Download*).

If the location of this application server is *In Build*, you can find it in `$TGH$/bundled` (for example, SunEE Reference Implementation J2EE or Tomcat).

If the location is *Download*, you can download this application server using the corresponding hyperlink.

Generic server options

You can generate EJB 1.0, EJB 1.1, or EJB 2.0 compatible deployment descriptors without choosing a specific application server. To do this, select Generic 1.0, Generic 1.1, or Generic 2.0 as the application server.

When the target application server is generic, Together creates the deployment descriptor for `ejb-jar.xml` according to the respective EJB specification. Together adds information about references or relationships satisfying the corresponding EJB specification to the deployment descriptor.

If you choose Generic 1.0, the J2EE Deployment Expert can generate:

- `ejb-jar.xml` for EJB modules
- `web.xml` for Web applications

If you choose Generic 1.1, 2.0, the J2EE Deployment Expert can also generate:

- `ejb-jar.xml` for EJB modules
- `web.xml` for Web applications
- `application.xml` for enterprise applications

For Generic 2.0 you can import existing JAR files prepared according to EJB 2.0 specification.

For Generic x.x, there are plug-ins for the following operating systems: Windows 98, Windows 2000, Windows Me (Windows Millennium Edition), Sun OS (Solaris), HP Unix, OSF/1 (Open Software Foundation), Linux, Macintosh OS (Mac OS X).

Installations and resources required for deployment

The following installations and permissions are prerequisites to deploying:

- An installation of Together with full deployment support.
- An accessible installation of the target application server with appropriate access rights.
- An accessible installation of Java Enterprise Edition 1.2 or higher.
- An accessible installation of Java 2 SDK (JDK 1.2 or higher).
- An accessible temp directory and sufficient disk space for temporary files that are generated by the J2EE Deployment Expert.

- An http-accessible location on the server for generated JSPs (if generating a JSP test client).

Important The Java Virtual Machine invokes the Together built-in compiler on the same JVM where JARs are placed. As a result, JAR files might be locked by Together after compilation. You can avoid this situation by using an external compiler (set up the external compiler in the Options dialog). For a complete description of the problem, refer to the *Readme.html* file in the root of your Together installation.

Together provides the following plug-ins for starting application servers:

- Start SAP J2EE AS 6.20
- Start Oracle 9i Application Server 1.0.2.2.1
- Start Sun ONE Application Server 6.5
- Sun EE Starter (built-in SunEE Reference Implementation J2EE 1.3.0 Application Server)
- WebLogic Server 5.1.0 Starter
- WebLogic Server 6.0 Starter
- WebLogic Server 6.1 Starter

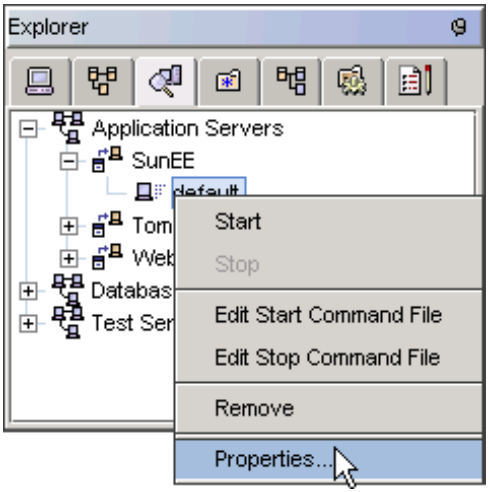
You can start these application server from the Explorer pane or from the J2EE Deployment Expert.

Starting application servers from Together

To start an application server from the Explorer pane:

1. In the Explorer pane, open the Server Explorer tab . Make sure that all properties of your server are defined correctly. For example, to view or edit the SunEE reference implementation server properties, navigate to *Application Servers - Sun EE - default*. Right click and choose **Properties**, as shown in Figure 97.

Figure 97 Accessing the SunEE reference implementation properties.



2. If necessary, edit the *Start command file* for your server. Right-click the server in the Explorer pane and choose **Edit Start Command File**.

   Edit example: For BEA WebLogic 6.1, you should add two lines to the command file `startWebLogic.cmd`:

   1. In the beginning of this file after the `SETLOCAL` program line, and before the line `cd ..\.. program`, add this:

      ```
      cd D:\bea\wlserver6.1\config\mydomain\
      ```

      where `D:\bea\wlserver6.1` is the full path to the directory where BEA WebLogic 6.1 is installed. The result should look like this:

      ```
      SETLOCAL
      D:

      cd D:\bea\wlserver6.1\config\mydomain\
      cd ..\..
      ```

   2. Set your BEA WebLogic password, so that you will not be prompted for the password during server startup. Add a new the line before `set STARTMODE=true`. The result should look like this:

      ```
      set WLS_PW=together
      set STARTMODE=true
      ```

Note On UNIX systems, `JAVA_HOME` might not be set in `StartWeblogic*.sh`. In this case, set `JAVA_HOME` in the script manually.

3. If necessary, edit the *Stop command file* for your server. Right-click the server in the Explorer pane and choose **Edit Stop Command File.**
4. To launch your server from Together, right click the server in the Explorer pane and choose **Start**.

An alternative method to start an application server:

1. Open a project.
2. Open a diagram appropriate for deployment (see "Diagrams for deployment" on page 404).
3. From the main menu, choose **Deploy | J2EE Deployment Expert**.
4. The first page of the Expert has a drop-down list of application servers with the currently integrated server platforms. Choose an application server to start from the available plug-ins listed at the beginning of this section.

J2EE Deployment Expert features

The J2EE Deployment Expert guides you through the deployment process. This section discusses the general features of the Expert.

Diagrams for deployment

You can invoke the J2EE Deployment Expert from diagrams of the following types:

- Class diagrams (creates a JAR archive file).
- EJB assembler diagrams (creates a JAR archive file). See Chapter 21, "Assembling EJB Modules" on page 337 for more information.
- Web application diagrams (creates a WAR archive file). See Chapter 23, "Designing and Developing Web Applications" on page 359 for more information.
- Resource adapter diagrams (creates an RAR archive file). See "Designing and Developing Resource Adapters" on page 393"Designing and Developing Resource Adapters" for more information.
- Enterprise application diagrams (creates an EAR archive file). See "Visual assembly of enterprise applications for deployment" on page 379 for more information.

If your application contains multiple EJB assembler diagrams and Web application diagrams, you can create an enterprise application diagram to map your existing modules.

Specifying parameters

The J2EE Deployment Expert enables you to specify the following:

- The target application server platform.
- Which deployment-related actions you want to take place (such as compile).

- Paths to the server, server tools, and the deployment output.
- Connection parameters for the application server.
- Optional generation of a simple JSP client for live-testing the deployed EJBs.
- Optional verification and correction.

Running the J2EE Deployment Expert

You can start the J2EE Deployment Expert from a diagram that supports deployment (class, enterprise application, EJB assembler, resource adapter, and Web application).

To run the J2EE Deployment Expert:

1. Open a diagram that supports deployment.
2. Choose **Deploy | J2EE Deployment Expert** on the main menu.
3. On the first page of the Expert, the drop-down list of application servers shows the server platforms currently integrated. Generic 2.0 application server is set by default. Select the application server that you want to target.

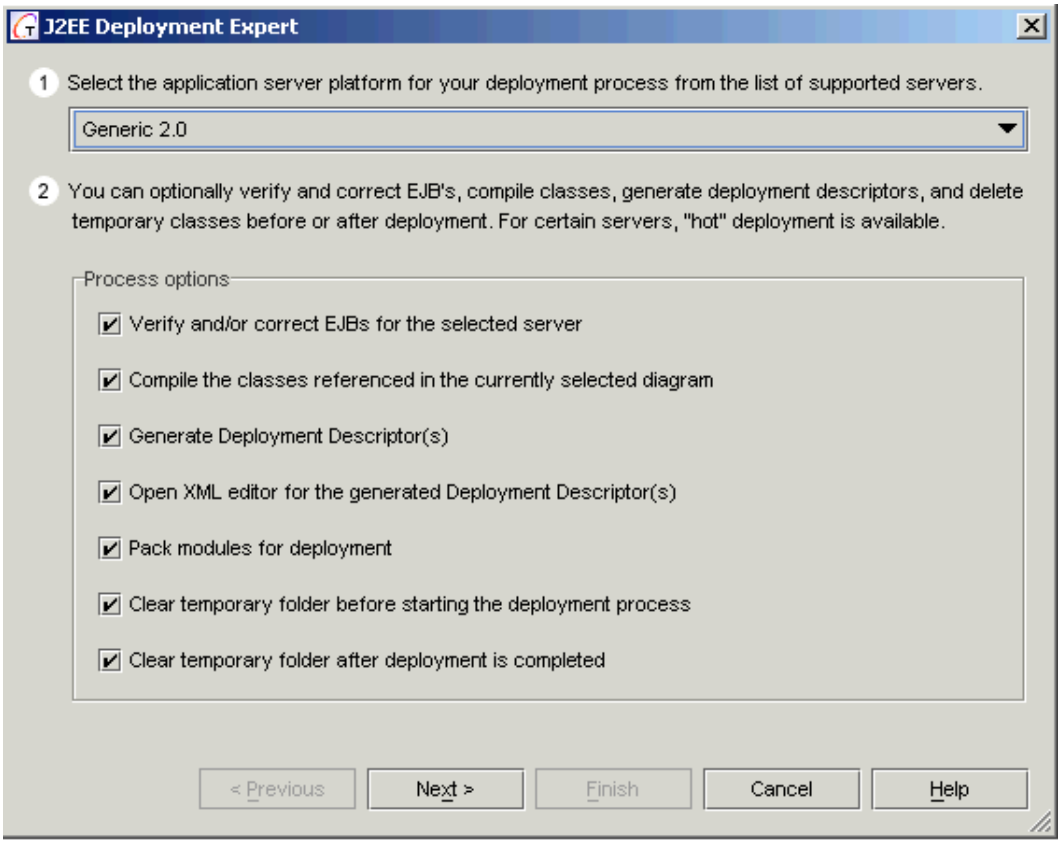
   Figure 98 shows the first page of the J2EE deployment Expert for the Generic 2.0 specification for an application server.
4. Go through the pages of the Expert, filling in the options as desired. Use **Previous** and **Next** buttons to navigate through the Expert's pages.
5. Click **Finish** to start the deployment process.

As its initial step, Together tests the project for EJBs or other elements that should be deployed to an application server.

According to your options, Together interacts with the compiler to compile EJB or other classes, verifies the EJB classes, generates the XML deployment descriptors, updates the EJB specification conformity, generates container classes, and then packages everything into a JAR (WAR, RAR, or EAR) file, and deploys to the location you specify. On server platforms that support it, you can specify "hot" deployment directly to the application server.

Important If the deployment fails unexpectedly, a possible reason is that the target application server does not support the current encoding. See "Encoding in deployment descriptors" on page 430 for more information.

Figure 98 Default *Process options* page


J2EE Deployment Expert
1 Select the application server platform for your deployment process from the list of supported servers.
Generic 2.0
2 You can optionally verify and correct EJB's, compile classes, generate deployment descriptors, and delete temporary classes before or after deployment. For certain servers, "hot" deployment is available.
Process options
Verify and/or correct EJBs for the selected server
Compile the classes referenced in the currently selected diagram
Generate Deployment Descriptor(s)
Open XML editor for the generated Deployment Descriptor(s)
Pack modules for deployment
Clear temporary folder before starting the deployment process
Clear temporary folder after deployment is completed
< Previous
Next >
Finish
Cancel
Help

Main scenarios for using the J2EE Deployment Expert

There are two main scenarios for using the J2EE Deployment Expert:

- For “fast track” deployment with default values for security and permissions or for deployment prototyping, start the J2EE Deployment Expert from a class diagram. Choose the target server platform and set the other options as desired.
- For “full featured” assembly information with control over security and permissions, start the J2EE Deployment Expert from an EJB assembler diagram, Web application diagram, resource adapter diagram, or enterprise application diagram.

Web application and enterprise application diagrams can be used with these choices of target application servers: Generic 1.1, Generic 2.0, IBM WebSphere 4.0, BEA WebLogic 6.0, 6.1 and 7.0, Sun ONE 6.5, Borland Enterprise Server 5.2, and SunEE 1.2, 1.3.

Configuration files

Deployment is possible for different operating system platforms using special plug-in config files. If you want to change something in the one of plug-in files, consider these naming rules:

- If no OS-specific `config` file is defined, the common `<plug-in-name>.config` file is used.
- OS-specific `config` files have the suffix `java.lang.System.getProperty ("os.name")` with all non alpha-numeric symbols replaced by underscores. For example, if you have `MYASPlugin`, then `configs` would be:

  `MYASPlugin.config`: Common `config` for all "unknown" operating systems

  `MYASPlugin.config_Windows_NT`: Main `config` for Windows NT

  `MYASPlugin.config_SunOS`: Main `config` for Solaris

  `MYASPlugin.config_Linux`: Main `config` for Linux

  `MYASPlugin.settings.config_Windows_NT`: User settings specified in last session of the deployment.

Server-specific deployment information

This section provides server-specific information for starting deployment on the Sun EE reference implementation application server, BEA WebLogic application servers, and IBM WebSphere application servers.

Note This User Guide assumes that you are familiar with the requirements of your application server.

You should make sure that all the necessary libraries are added to your project. Together automatically adds the maximum number of available libraries in the plug-ins, but some servers require that you add paths. For example, according to SunEE requirements, you should add the `client.jar` to the project class path before running to avoid an exception. If the EJBs and other elements are deployed in different projects, you should add the paths to these archive files to the project.

Sun EE reference implementation application server

Together provides the Sun EE reference implementation J2EE 1.3.0 application server within the Together build. This application server is a default implementation that is helpful for conceptual testing.

Together supports the Sun EE Reference Implementation for J2EE 1.2.1 application server as well the built-in version. If you want to use Sun EE Reference Implementation J2EE 1.2.1, you should first install it.

Note All projects deployed to Sun EE Reference Implementation J2EE 1.2.1 deploy successfully to Sun EE Reference Implementation J2EE 1.3.0.

Borland Enterprise Server 5.2

Together supports the deployment to Borland Enterprise Server 5.2. To hot deploy from Together, first start this application server.

Note Together currently supports deployment to BES 5.2 for Windows and Solaris only.

IBM WebSphere Application Server

Together supports the IBM WebSphere 3.02, 3.5, and 4.0 application servers. Together supports both IBM WebSphere 4.0 Advanced Single Edition and IBM WebSphere 4.0 Advanced Edition.

Important When you are developing the project for IBM WebSphere application servers, make sure that you create a package. IBM WebSphere does not work without packages.

With IBM WebSphere 3.5 and 4.0 application servers, you can initiate the deployment process from class, EJB assembler, Web application and enterprise application diagrams. You can also place shortcuts to application client diagrams on enterprise application diagrams for subsequent deployment to IBM WebSphere 4.0.

BEA WebLogic Application Server

Together supports BEA WebLogic 4.5.1, 5.1, 6.0, 6.1, and 7.0 Application Servers.

Note BEA WebLogic 6.1 and 7.0 application servers support EJB 2.0 specification. You can use applications previously developed under EJB 1.1 specification without any code modification (this is very important for CMP Entity Beans).

Important Be careful when deploying a project satisfying the EJB 2.0 specification to BEA WebLogic Application Server 6.0, which supports only public draft 2 of EJB specification 2.0. (In that draft, EJB relationships were not completely implemented.) However, the Together plug-in for BEA WebLogic 6.1 completely supports EJB relationships in accordance with the final release of EJB specification 2.0.

You can deploy to BEA WebLogic 6.1 and 7.0 from any of the diagrams listed in "Diagrams for deployment" on page 404. You can deploy to BEA WebLogic 6.0 for any of those diagrams except resource adapter diagrams.

You can deploy to BEA WebLogic 5.1 from class diagrams or EJB assembler diagrams only (JAR archive file is created). You can also generate a deployment descriptor using a Web application diagram.

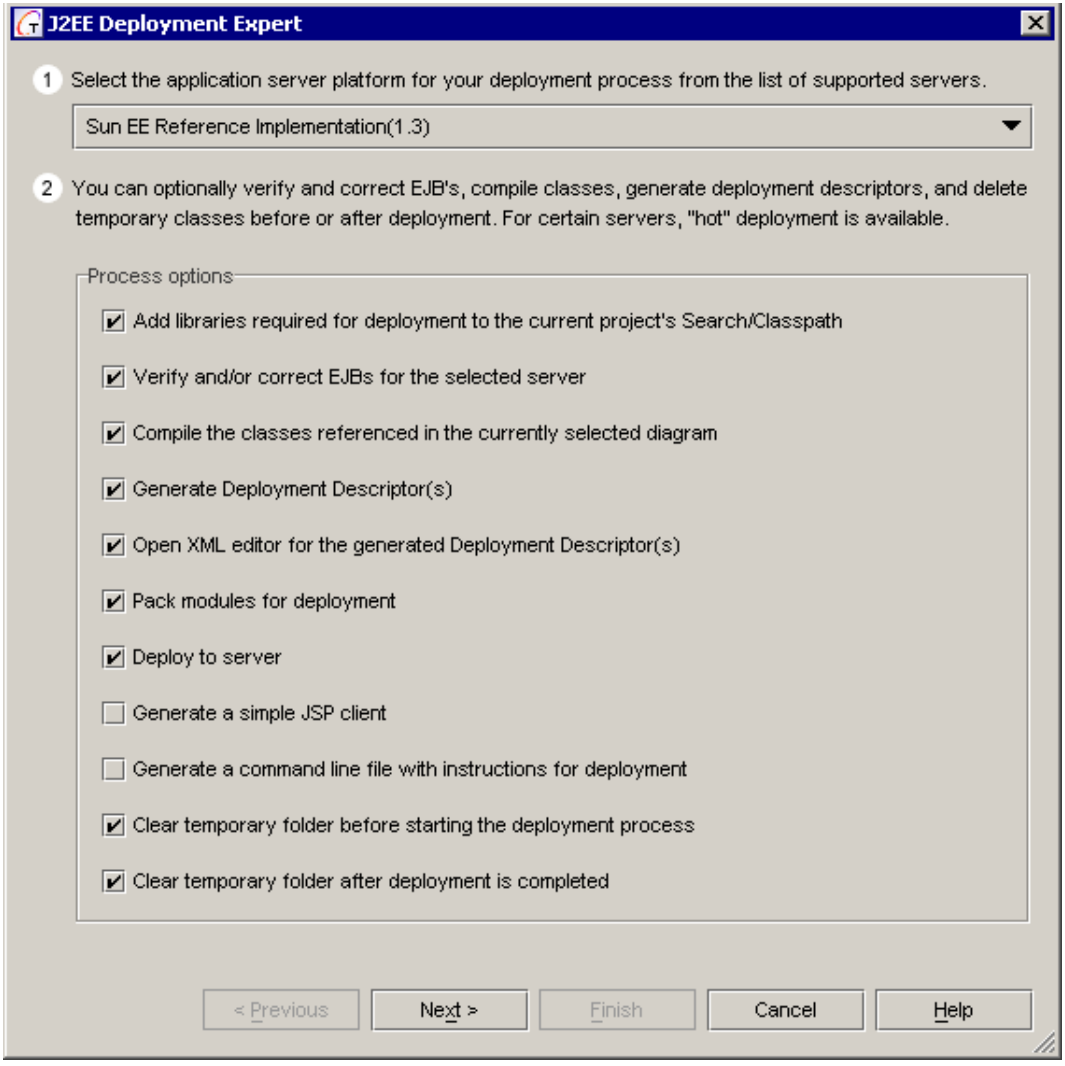
J2EE Deployment Expert options common to all servers

The J2EE Deployment Expert is a guide that takes you through several pages of options that describe deployment tasks and settings. The pages depend on the current application server. This section discusses the options on pages that are common to a few application servers: Sun EE Reference Implementation 1.2.1, 1.3.0, IBM WebSphere 3.5 and 4.0, and BEA WebLogic 5.1.0, 6.0, 6.1, and 7.0.

First page: process options

The first page of the Expert displays the process options. Figure 99 shows the process options for the Sun EE reference implementation application server.

Figure 99 Process options for Sun EE reference implementation


J2EE Deployment Expert
1 Select the application server platform for your deployment process from the list of supported servers.
Sun EE Reference Implementation(1.3)
2 You can optionally verify and correct EJB's, compile classes, generate deployment descriptors, and delete temporary classes before or after deployment. For certain servers, "hot" deployment is available.
Process options
Add libraries required for deployment to the current project's Search/Classpath
Verify and/or correct EJBs for the selected server
Compile the classes referenced in the currently selected diagram
Generate Deployment Descriptor(s)
Open XML editor for the generated Deployment Descriptor(s)
Pack modules for deployment
Deploy to server
Generate a simple JSP client
Generate a command line file with instructions for deployment
Clear temporary folder before starting the deployment process
Clear temporary folder after deployment is completed
< Previous
Next >
Finish
Cancel
Help

The options in the Expert that common to Sun EE, Borland Enterprise Server 5.2, BEA WebLogic, and IBM WebSphere are as follows:

- **Add libraries required for deployment to the current project's Search/ Classpath:** EJB deployment requires several server-specific libraries. Because diverse servers use different versions of the J2EE specification, server vendors generally suggest adding their own libraries that cover the J2EE specification. (For Sun EE deployment, Sun EE is added to the classpath. For Borland Enterprise Server 5.2, this option is absent).

Note If all the necessary libraries have already been added to the classpath, you should uncheck this option to avoid problems with the compiler locking JAR files.

- **Verify and/or correct EJBs for the selected server:** Verifies that the classes on the current diagram correspond to the EJB specification supported by the target server (plus individual specifications for some servers). The primary objective of this task is to make sure that all the required classes exist on the diagram. The current diagram is used as the source of the EJBs.

Note If you have both EJBs supporting EJB 1.1 specification (including CMP EJB1.1) and EJBs supporting EJB 2.0 specification in the same project, uncheck **Verify and/or correct EJBs for the selected server**.

- **Compile the classes referenced in the currently selected diagram:** Compiles the bean, packs it in a JAR file, and stores the JAR in a temporary folder. **Together** uses the standard Sun tools (`javac.exe` and `jar.exe`) for compilation and packing.
- **Generate Deployment Descriptor(s):** Generates deployment descriptors. As soon as the descriptors are ready, they are added to the JAR, WAR, RAR, or EAR archive together with some additional classes.

Note **BEA WebLogic.** Together uses the BEA WebLogic tool `ejbc.exe`.

- **Pack Modules for Deployment:** Creates deployable JAR, WAR, RAR, or EAR archives.
- **Generate a simple JSP client:** Generates a JSP client for the deployed EJBs. The client is a set of interrelated JSP and HTML files, which you can view in a browser. The intention of this client is to demonstrate access to remote EJB objects through an open interfaces. The semantic information required for JSP client generation is taken from the active diagram of the current project. Together provides this feature as a universal testing facility .

  It is possible to generate a JSP client from Web application and enterprise application diagrams.

Note Since EJBs reside inside the container and JSPs typically reside outside, the generated JSP client does not support EJBs that do not have remote or home interfaces (it ignores local interfaces). The JSP client does not support message-driven beans either.

- **Generate a command line file with instructions for deployment:** Generates a command file consisting of the sequence of calls to various console tools for the application server. You can subsequently launch this file to perform all actions required for deployment.
- **Clear temporary folder before starting the deployment process / Clear temporary folder after deployment is complete:** Uncheck these options if you want to reuse temporary files.

Sun EE and Borland Enterprise Server 5.2 process options

There is one process option for Sun EE and Borland Enterprise Server 5.2 in addition to the common process options listed above:

- **Hot Deploy to server** or **Deploy to server:** Does the hot deploy. This step makes use of a bean already compiled and jarred, so you must use this step in conjunction with the others. The server must already be started.

You should compile classes, generate descriptors, and deploy the beans all together, because some tasks use the results of the previous tasks. For example, hot deploy requires a `*.jar` file to be prepared in a temporary folder.

IBM WebSphere process options

Process options for IBM WebSphere 4.0 are similar are similar to those for IBM WebSphere 3.5. Several options are in addition to the common ones listed earlier.

- **Enable Advanced Single Edition support:** (IBM WebSphere 4.0 only) Check to use the Advanced Single Edition variant of IBM WebSphere 4.0. Do not check this option if you are going to use the Advanced Edition version.
- **Deploy to server:** Deployment executes immediately. WebSphere must already be started. This step makes use of a bean that has already been compiled and jarred.

Note IBM WebSphere 4.0 Advanced Single Edition does not support hot deployment. You should restart the application server. IBM WebSphere 4.0 Advanced Edition and IBM WebSphere 3.5 do support hot deployment. For IBM WebSphere 3.5, the name of this option is **Hot Deploy to server**.

- **Generate executable files for the client:** (IBM WebSphere 3.5 only) Together and WebSphere use different versions of the JDK. WebSphere requires using the IBM JDK for client programs. Thus you cannot make use of Together internal tools. You must generate two batch files, one to compile and the other to run the client program. Subsequently, you can start them separately from Together.
- **Process Servlet(s):** (IBM WebSphere 3.5 only) Check this box to deploy servlets.
- **Launch J2EE Application Client:** (IBM WebSphere 4.0 only) Launches the already deployed application client.

  If you use an application client diagram to create your application client, place the shortcut for the application client diagram on the current enterprise application diagram and then deploy.

BEA WebLogic process options

The process options for BEA WebLogic 5.1, 6.0, 6.1, and 7.0 include all of the common options listed earlier as well as an additional one:

- **Hot Deploy to server:** Does the hot deploy. This step makes use of a bean already compiled and jarred, so you have to use this step in conjunction with the others. The server must already be started.

You should compile classes, generate descriptors, and deploy the beans at the same time, because some tasks use the results of the previous tasks. For example, hot deploy requires a `*.jar` file to be prepared in a temporary folder.

Note All hot deployed beans and deployment descriptors are stored in the Together temporary folder and in the WebLogic home folder. You can also save these files in a special folder. If you choose BEA WebLogic 7.0 as a target application server, Together generates a correct `web.xml` file according to the Java Servlet specification, Version 2.3.

The following options are available for the BEA WebLogic 4.5.1 application server:

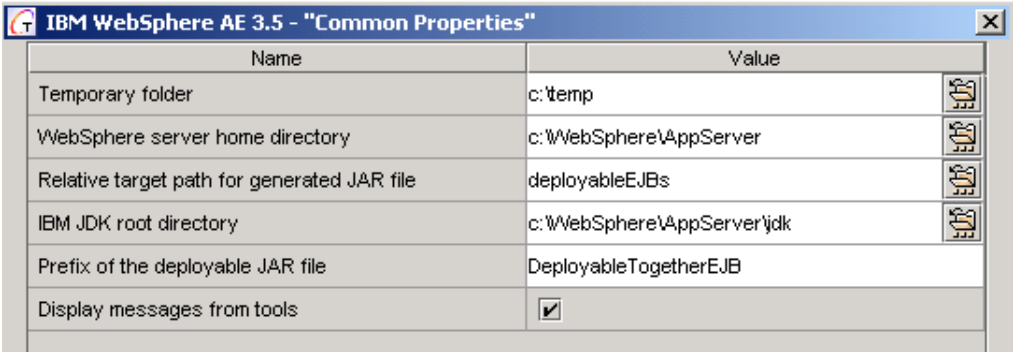
- Verify and/or correct EJBs for the selected server
- Compile the classes referenced in the currently selected server
- Generate Deployment Descriptor(s)
- Hot Deploy to server
- Clear temporary folders before starting the deployment process
- Clear temporary folders before/after deployment is completed

The Common Properties page has fields for locations of folders, the JDK, and servers. For every application server, this page has at least three fields:

- **Temporary folder:** Location of temporary files generated during deployment.
- **Folder for the generated JAR and EAR files** *or* **Relative target path for generated JAR file** (WebSphere 3.5)**:** Location of the resulting JAR file and where it will be loaded by the server.
- **<Server> root or home directory:** Location of the server.

Figure 100 shows the common properties page for IBM WebSphere 3.5.

Figure 100 Common properties for IBM WebSphere 3.5



Sun EE common properties

The special Sun EE common properties include the following:

- **JDK 1.3 root directory:** Location of the JDK.
- **J2EE root directory:** Location of the directory `$TGH$/bundled/j2ee`.

Borland Enterprise Server 5.2 common properties

The special Borland Enterprise Server 5.2 common properties include the following:

- **JDK 1.4 root directory:** Location of the JDK.
- **Server root directory:** Location of the Borland Enterprise Server 5.2 directory.

IBM WebSphere common properties

Common properties for IBM WebSphere include the following:

- **IBM JDK root directory:** (WebSphere 4.0) Location of JDK 1.3.
- **Prefix of deployable JAR file**: (WebSphere 3.5) The name assigned to the JAR file (or files) generated in the *Compile Classes* task from the selected diagram. While preparing for deployment, Together creates an additional JAR file whose name has this prefix.
- **Display messages from tools:** (WebSphere 3.5 only) Displays tool messages in the Message pane.

BEA WebLogic common properties

The BEA WebLogic common properties has one field in addition to the common ones listed above.

- **JDK 1.3 root directory:** The JDK location.

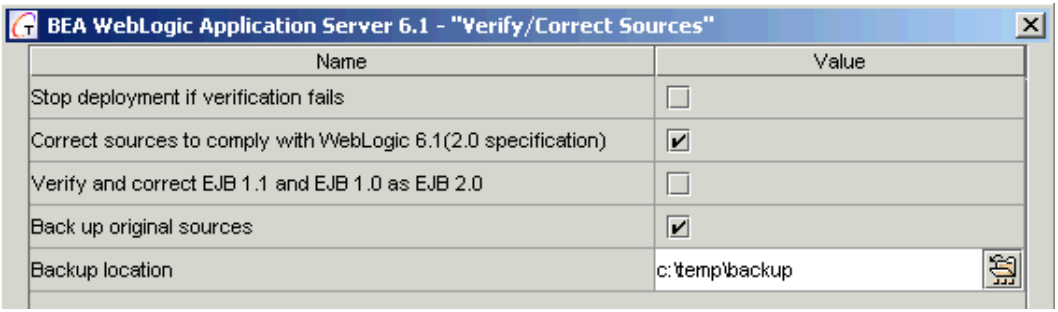
Important Set paths to jdk1.2 or later for BEA WebLogic 5.1, and to jdk1.3 for BEA WebLogic 6.0 and 6.1. You cannot use jdk1.2 for BEA WebLogic 6.0, 6.1, or 7.0.

Verify / Correct Sources page

The Verify/Correct Sources page has fields for terminating deployment if verification fails, correcting sources to make them compliant with the application server, and making backup copies. Figure 101 shows the Verify/Correct Source page for BEA WebLogic 6.1.

You should use the fields on this page to provide verification or correction of the source Java code.

Figure 101 Verify/Correct page for WebLogic 6.1 application server


BEA WebLogic Application Server 6.1 - "Verify/Correct Sources"
Name
Value
Stop deployment if verification fails
Correct sources to comply with WebLogic 6.1(2.0 specification)
Verify and correct EJB 1.1 and EJB 1.0 as EJB 2.0
Back up original sources
Backup location
c:\temp\backup

Options for verification and correction include the following:

- **Stop deployment if verification fails**
- **Correct source code to comply with server specification:** Together can change code as needed.

Note IBM WebSphere 3.5 supports the EJB 1.0 specification. Later specifications require that any create methods of a home interface should return the primary key type, while the 1.0 specification requires that create methods return void. If you deploy a project originally targeted to a later specification to WebSphere 3.5, Together replaces all changed types with void.

- **Verify and correct EJB 1.1 and EJB 1.0 as EJB 2.0:** (BEA WebLogic and Sun EE only) If there are EJBs satisfying the EJB1.1 specification and EJBs satisfying the EJB 2.0 specification in the same project, all EJBs are deployed to the application server without any correction of the code.

Note **Sun EE.** If your project has EJBs satisfying different EJB specifications (EJB 1.0, 1.1, 2.0), you can deploy it to Sun EE Reference Implementation without verification and correction. There is only one exception: you cannot deploy CMP bean 1.0 together with EJBs satisfying EJB 1.1, 2.0 specification.

- **Back up original sources:** Backs up the original sources when Together changes the source code to comply with the server specification.
- **Backup location:** Directory for backup copies of the source code.
- **Uninstall already deployed application:** Check if you are going to deploy an application with the same name as an existing application deployed to the same application server.

Simple JSP Client Generation page

The Simple JSP Client Generation page displays only if you check *Generate a simple JSP client* on the first page of the Expert. Figure 102 shows the Simple JSP Client Generation page for BEA WebLogic 5.1.

These are the common entries for all servers:

- **Show JSP client in default Internet browser:** Launches your default Internet browser and displays the start page.
- **Root path for JSP storage:** The root of the file hierarchy, which can be accessed externally. You must place any HTML or JSP files that you want to access under the document root.

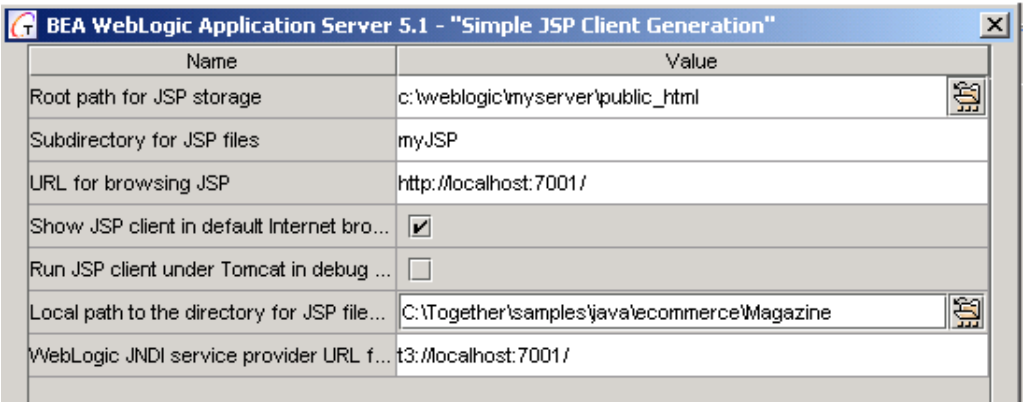
Note By default, the WebLogic public directory resides at `$WebLogic_home$/myserver/public_html`

Options for WebSphere 3.5

The additional JSP generation fields for WebSphere 3.5 are as follows:

- **Subdirectory for JSP files:** Subdirectory of the public directory for the JSP files. This path is relative to the root public directory. If the subdirectory does not exist, it is created automatically (after confirmation).
- **URL for browsing JSP:** During request processing, the server substitutes this base URL to the WebSphere public directory in order to access the requested Web resource. An invalid value in this field or the public directory causes an access error. By default, the base URL is `http://localhost`.
- **WebSphere JNDI service provider URL:** (WebSphere 3.5 only) Establishes a connection between a JSP client and an EJB server. The value in this field is inserted in a JSP file during generation and is subsequently used during execution. Normally, you do not need to change this field. The default value is `iiop://localhost:900`.

Figure 102 Simple JSP Client Generation page for BEA WebLogic 5.1



Options for WebLogic 5.1

The JSP client options for WebLogic 5.1 include these two, which are equivalent to their WebSphere counterparts listed above.

- **Subdirectory for JSP files**

- **URL for browsing JSP:** By default, the base URL is `http://localhost:7001`.

WebLogic 5.1 has additional options for running under debug mode.

- **Run JSP client under Tomcat in debug mode:** The generated JSP client runs under Tomcat rather than the BEA WebLogic server. In this case, JSPs interact with an EJB remotely. For this purpose, the source code of these JSPs contains lines for initialization of the Initializing object. The initialization text is stored in a configuration file and can be modified if necessary.

  If you check *debug mode,* the Navigation Page of the JSP client runs automatically. In this case, the *Show JSP client in default Internet browser* option is ignored.

- **Local path to the directory for JSP files for Tomcat:** Folder of the generated JSP files for running under Tomcat. The folder is an alternative to the one in the WebLogic public directory. Its default value is the root directory of the current project.

- **WebLogic JNDI service provider URL for Tomcat:** The value of the URL is used for creating parameters for the `InitialContext` object. This helps avoid making manual changes to the configuration file if the BEA WebLogic server is remote or uses a different port number.

Options for Sun EE, Borland Enterprise Server 5.2, WebSphere 4.0, WebLogic 6.0, 6.1, 7.0

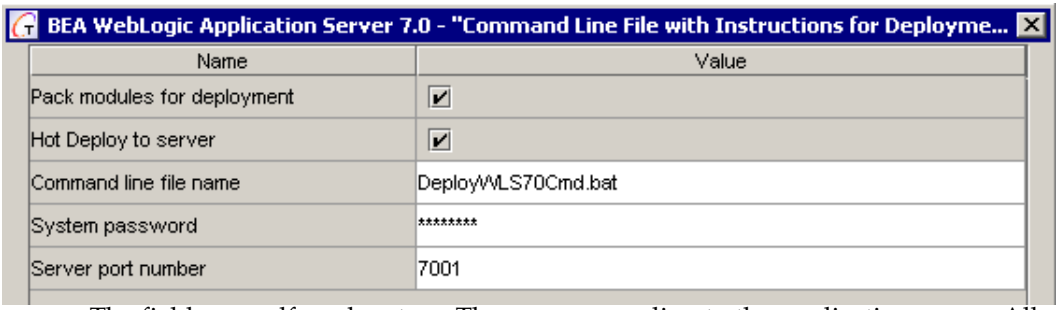
The following options are in addition to the ones common for all servers.

- **Name of the target Web Application Diagram:** The Web application diagram where Together will store the generated JSPs.
- **Open target Web Application Diagram:** Opens the diagram named above.
- **Clear target directory before deploying**
- **Web server host name:** Default value for WebLogic is `localhost`.
- **Web server port number:** Default value for WebLogic is `7001`.

Command Line File with Instructions for Deployment page

This page displays if you check the box *Generate a command line file with instructions for deployment* on the first page of the Expert. Figure 103 shows this page for BEA WebLogic 7.0 application server.

Figure 103 Command Line File page for WebLogic 7.0



The fields are self-explanatory. They vary according to the application server. All of them however, have this field:

Command file name: Name of file with deployment instructions.

When you check an option in this page, Together adds a command for the associated task in the command line file.

Server-specific J2EE Deployment Expert pages

Pages of the J2EE Deployment Expert vary according to the application server. The previous section discussed pages and options common to several servers. This section discusses the pages that vary among servers according to title and intent or according to fields.

Sun EE Deployment Properties page

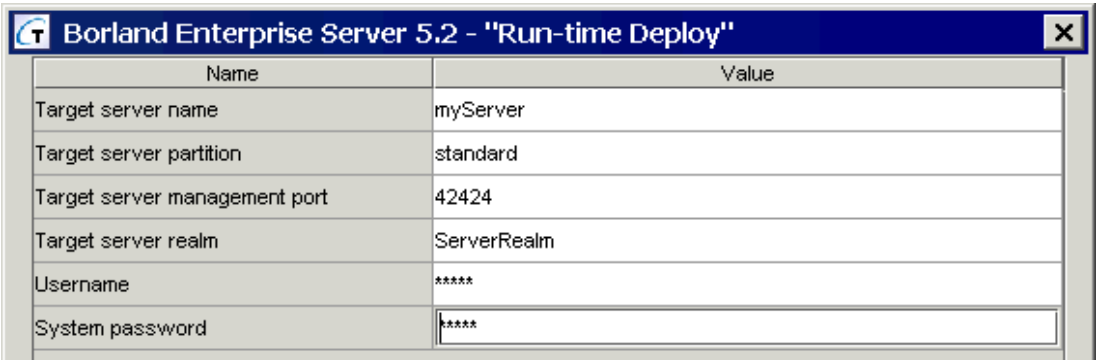
The Deployment Properties page for Sun EE has information necessary for Together to communicate with the application server. The fields are as follows.

- **Server host name:** Default is `localhost`.
- **Web server port number:** Default for Sun EE is `8000`.
- **Uninstall already deployed application:** Check if you are going to deploy an application with the same name as an existing application already deployed.

Borland Enterprise Server 5.2 Run-time Deploy properties page

The Run-time Deploy properties page has information required for hot deployment. Figure 105 shows this page for Borland Enterprise Server 5.2.

Figure 104 Run-time Deploy page for Borland Enterprise Server 5.2



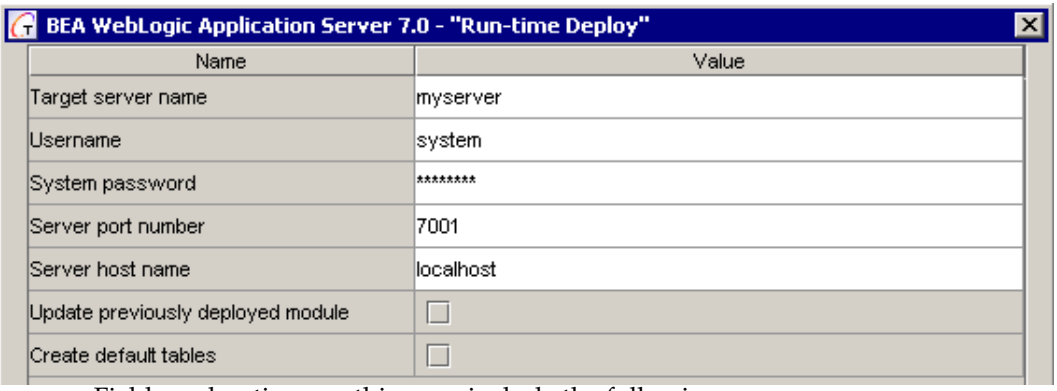
Fields and options on this page include the following:

- **Target server name:** the name of Borland Enterprise Server 5.2.
- **Target server partition:** Default is `standard`.
- Target server management port: The default server port is 42424. If you do not know the server port number, you can find it in the Borland Management Console.
- Target server realm: Default is `ServerRealm`.
- **Username:** Username for your Borland Enterprise Server 5.2 installation. Default is `admin`.
- **System password:** Defined during Borland Enterprise Server 5.2 installation.Default is `admin`.

BEA WebLogic Run-time Deploy properties page

The Run-time Deploy properties page has information required for hot deployment. Figure 105 shows this page for BEA WebLogic 7.0 applications server.

Figure 105 Run-time Deploy page for BEA WebLogic 6.1



Fields and options on this page include the following:

- **Target server name:** WebLogic server name.
- **Username:** Username for your WebLogic installation. Default is `system`.
- **System password:** Defined during BEA WebLogic installation.
- **Server port number:** The default server port is 7001. If you do not know the server port number, you can find it in the file:

  `weblogic.property` for BEA WebLogic 5.1.0

  `config.xml` for BEA WebLogic 6.0 and 6.1
- **Server host name:** By default, the server host is `localhost`. Because the Expert uses the file system access to BEA WebLogic server, the server runs on the same computer.
- **Update already deployed module:** During run-time deployment to BEA WebLogic, it is possible to update any modules that have already been deployed. Check this box to update modules during deployment.
- **Create default tables:** (WebLogic 7.0 only) Corresponds to the deployment descriptor tag `<create-dbms-default-tables>`.

Before proceeding to the next page of the expert, make sure that the server is available at the selected port and host and that the current application server has been started.

IBM WebSphere 3.5 pages

The IBM WebSphere 3.5 deployment options differ significantly from those for other servers.

EJB Deployment Properties page

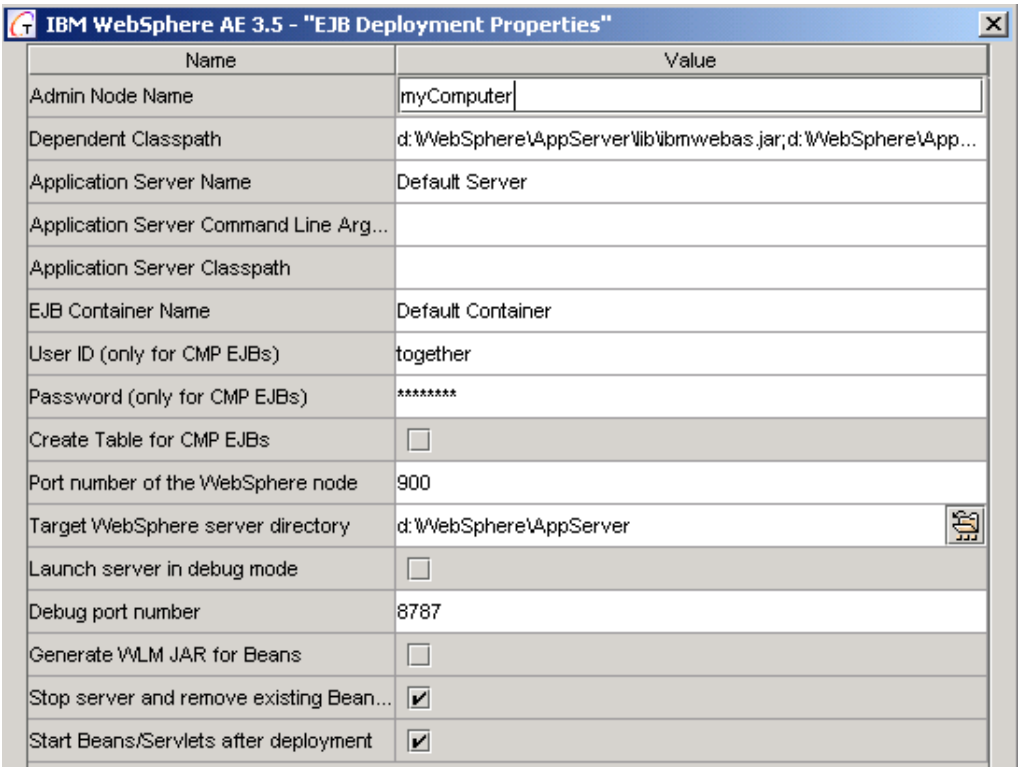
Figure 106 shows the EJB Deployment Properties page for IBM WebSphere 3.5, The deployment options include the following:

- **Admin Node Name:** Local computer name (by default, assigned automatically).
- **Dependent Classpath:** Path to source classes, service library jars, and so on.
- **(Application) Server Name:** A single WebSphere node can hold several virtual application servers. By default, only one is allocated and its name is Default Server. If you use a different application server, enter its name here. If the server with that name does not exist, it is created.
- **Application Server Command Line Arguments / Application Server Classpath:** Corresponds to the properties on the application server's Administrative Console.
- **EJB Container Name:** A single WebSphere virtual Application Server can hold several EJB containers. By default, only one is allocated and its name is Default Container. If you use a different container, enter its name in this field. If the container with this name does not exist, it is created.
- **User ID / Password:** (CMP beans only) You can use the user name and password assigned during WebSphere installation.
- **Create Table for CMP EJBs:** (CMP beans only) Creates new tables in the server persistence repository designed to support Container Managed Persistence. The structure of the tables corresponds to the beans being deployed.
- **Port number of the WebSphere node:** Port for communication between the application server and client program. It is assumed that the host is `localhost`. Before proceeding with the next page of the J2EE Deployment Expert, make sure that the server is available for the selected port and host.
- **Target WebSphere server directory:** Path to the WebSphere application server installation root.
- **Launch server in debug mode:** Runs the server in debug mode and attaches to it from the debugger process in Together. This option does not restart the server in debug mode. Instead, it adds special command-line arguments to the application. You must restart the application server from the WebSphere Administrative Console.

Note Several additional libraries are required to launch WebSphere in debug mode. See WebSphere InfoCenter documentation for details.

- **Debug port number:** Port number for communication between WebSphere and the external debugger. The server and debugger must use the same port number. You can use the default port number 8787.
- **Generate WLM Jar for Beans:** Creates a work load management file.

Figure 106 Deployment Properties page for IBM WebSphere 3.5



Servlet Deployment Properties page

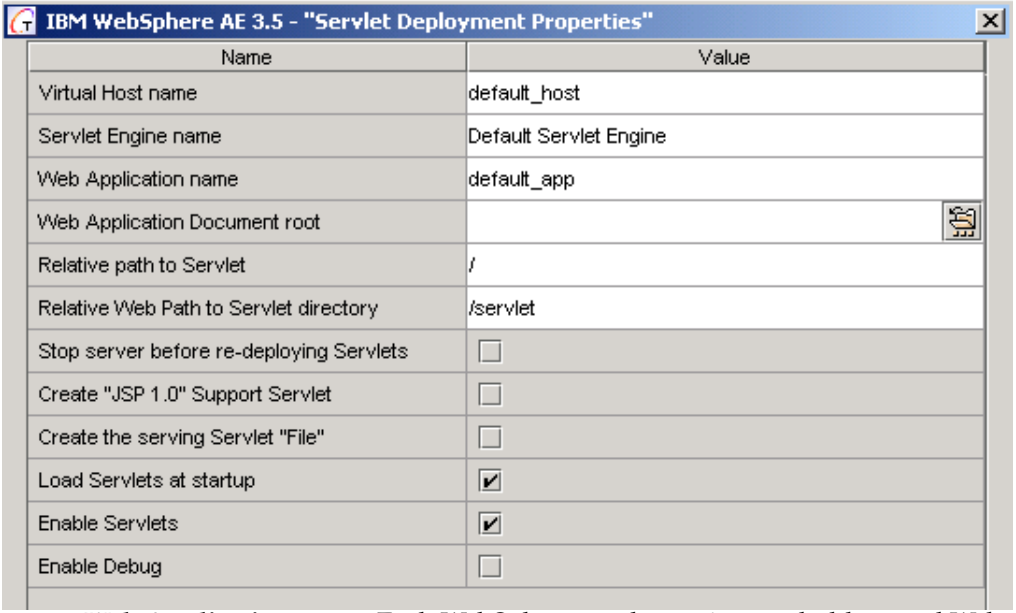
This page opens only if *Process Servlet(s)* is checked. You can use this page for creating servlets and WebSphere service servlets. Figure 107 shows the Servlet Deployment Properties page.

The properties on this page include the following:

- **Virtual Host name:** A single WebSphere Application Server can hold and serve several virtual Web hosts. Each virtual host is associated with a set of aliases. By default, only one host is allocated, and its name is `default_host`. It is associated with the `localhost`. To use a different virtual host, specify its name here.

- **Servlet Engine name:** Each WebSphere virtual Application Server can hold several servlet engines. By default, only one engine is allocated. Its name is Default Servlet Engine. If you use a different servlet engine, enter its name here. If the engine with the specified name does not exist, it is created.

Figure 107 Servlet Deployment Properties for IBM WebSphere 3.5



- **Web Application name:** Each WebSphere servlet engine can hold several Web applications. The J2EE Deployment Expert uses the `default_app` Web application. If you are going to use a different Web application, enter its name here. If the application with the specified name does not exist, it is created.
- **Web Application Document root:** Each Web application on the WebSphere server has its own document root, which is a file system address accessible through an Internet browser. All subdirectories of the root are also accessible. Contents of this field should be equal to the corresponding property of the Web application in the WebSphere Advanced Administrative Console.
- **Relative path to Servlet:** Each Web application on the WebSphere server has its own Web path. This is a string similar to a directory path except that it does not refer to any actual directory; instead it is a Web resource alias for the Web application. The Web path is appended to the virtual host alias. For example, if the virtual host is `localhost:900` and the Web path is `/webapp/Examples`, then the result is `http//localhost:900/webapp/Examples`. If *Web Application* is empty, the Web path consists of a single slash character (/).

- **Relative Web Path to Servlet directory:** The part of Web path to access a servlet. This is appended to the relative path to servlet, described above. For example, if the virtual host is `localhost:900`, the Web path is `/webapp/Examples`, and the relative Web path to a servlet is `/servlet`, then the result will be `http//localhost:900/webapp/Examples/servlet`.

  If *Relative Web Path to Servlet directory* is empty, it consists of a single slash character (/).

- **Stop Server before re-deploying Servlets:** Stopping or restarting a running servlet requires Application Server Node shutdown, resulting in the loss of all deployed objects. Check this box if the running beans are of no interest, and restart the servlets together with the application server. If you preserve the running beans, the new servlets still deploy, but they do not stop or start regardless of the previously set options *Stop (remove) EJBs, Servlets,* and *Start EJB's, Servlets* on the page *EJB Deployment Properties*.

- **Create "JSP 1.0" Support Servlet:** If the project uses JSPs, this indicates whether a special servlet will be added to the Web application. The name of this servlet class is `com.sun.jsp.runtime.JspServlet`. In WebSphere it is normally called `jsp10`.

- **Create the serving Servlet "File":** If your project uses HTML resources, this flag indicates whether a special servlet responsible for processing requests to HTML resources will be added to the Web application. The name of this servlet class is `com.ibm.servlet.engine.webapp.SimpleFileServlet`. In WebSphere it is normally called `file`. This servlet can be added manually or by checking the box.

- **Load Servlets at Startup:** Loads a servlet when the application server starts rather than when the servlet is requested. (Default is unchecked.)

- **Enable Servlets:** Indicates whether the servlet is available to handle requests. (Default is checked.)

- **Enable Debug:** Indicates whether to start the servlet in Java debug mode. (Default is unchecked.)

Client Properties page

The Client Properties page contains properties required to generate start and compile files for the client task (if the corresponding box is checked on the first page of the Expert). Figure 108 shows the Client Properties page.

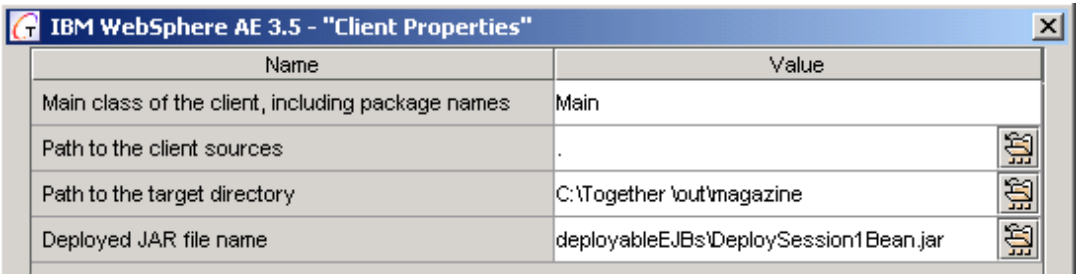
Properties on this page include:

- **Main class of the client, including package names:** Name of the class that contains the main method to run the client application. Automatically generated batch files refer to this class.

- **Path to the client sources:** Fully qualified path to the directory where client source files are stored.

- **Path to the target directory:** Fully qualified path to the directory where compiled files will be placed.

Figure 108 Client Properties page for WebSphere 3.5



IBM WebSphere 4.0 pages

Advanced Single Edition Properties page

The Advanced Single Edition Properties page opens only if the *Advanced Single Edition support* option on the first page of the Expert is checked. Figure 109 shows the two options on the page:

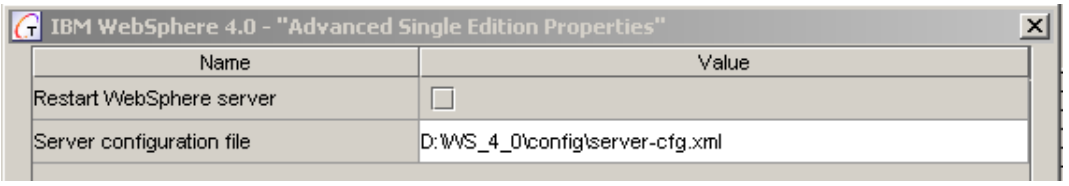
- **Restart WebSphere server:** Restarts IBM WebSphere 4.0 Advanced Single Edition.

Note IBM WebSphere 4.0 Advanced Edition supports hot deployment, but Advanced Single Edition does not support hot deployment. You should restart the current application server for deployment.

- **Server configuration file:** Saves information about EJBs in the corresponding server configuration file `server-cfg.xml`. Set the path to this configuration file.

Important IBM WebSphere 4.0 Advanced Single Edition application server does not verify the correctness of JNDI names for data sources and relationships with other beans. For this reason, a JAR file might contain incorrectly defined JNDI names.

Figure 109 *Advanced Single Edition Properties* page for IBM WebSphere 4.0



Deployment Properties page

The Deployment Properties page provides fields for hot deployment and other tasks. The options include:

- **Uninstall previous application:** Check this if there is an older deployed application with the same name on the current application server.
- **Install new application:** Check this if you are going to deploy a new application.
- **Node name:** Local computer name (by default, assigned automatically).
- **Server name:** Name of the current application server in the IBM WebSphere Administrative Console.
- **Virtual host name:** Name of the host in the IBM WebSphere Administrative Console.
- **JNDI name of the data source / Schema name:** Used for CMP beans only. JNDI name is the name used by the data source, and the Schema name is the name connected to your database. To fill in these fields you should start the IBM WebSphere Administrative Console and set the same settings as you have there.
- **User name and Password (for CMP EJBs):** Name and password defined during IBM WebSphere installation.

Making transitions among application servers

It is a common task to change application servers after you have implemented your project and deployed your application to an application server. This presents two major problems:

- Different servers support different EJB specifications (EJB 1.0, 1.1, 2.0).
- Each application server has its own specific features.

This section discusses transitioning among different EJB specifications and among different application servers.

Transitions among specifications

If you want to redeploy your application to a new application server, you should first determine which EJB specification the new server supports. If the new application server supports the same specification as the first server (for example, your project was implemented using EJB 1.1 and the targeted server also supports EJB 1.1), you need to consider only the specific features of the application server.

If the new application server does not support the EJB specification for the project's target server (for example, the new server supports a later EJB specification), you must correct the Java code of the EJB to meet the targeted EJB specification. The details of the Java code implementing the EJB component vary for different EJB specifications and different EJB components. Entity CMP beans require the most extensive code corrections to make a transition from one specification to another.

Together determines which EJB specification this application server supports, and verifies and corrects the EJBs according to the specification (EJB 1.0, 1.1, or 2.0).

Automatic verification and correction of EJBs

Together can automatically correct most of the EJB code that is specification sensitive.

To automatically verify and correct EJBs:

1. Open the diagram for the original deployment.
2. From the main menu, choose **Deploy | J2EE Deployment Expert**.
3. Choose the new application server from the list at the top of the Expert.
4. Check *Verify and/or correct EJBs*. You can clear all other options if you merely want to change the code without attempting to deploy.
5. Continue through the remaining pages as usual.

Together verifies and corrects only the methods described in the EJB x.x Specification. For an additional discussion of using verification and correction, see "EJB verification and correction" on page 343.

Manual correction of EJBs

Together cannot make all necessary corrections to your code, especially if you move from an earlier EJB specification to a later one. For example, if your EJB contains business methods (which are not included in the EJB specifications), Together verifies and "corrects" them if they do not satisfy the EJB specification. When Together corrects a business method, however, it replaces the body of the method with the default (empty) body.

Together saves a backup of all Java classes of the project in the backup folder. You can completely rewrite the code for a "corrected" business method, or simply use the appropriate fragments of your old code from the backup file.

Together corrects home and remote interfaces automatically. If you do not want to keep the corrections, you can use the backup code, manually making any necessary adjustments to meet the selected EJB specification.

Moving from EJB 2.0 to EJB 1.0

To understand how Together corrects code when you transition from the later EJB 2.0 specification to the earlier EJB 1.0 specification, consider a simple sample from BEA WebLogic 6.1 (EJB 2.0). Attempt the verification and correction by using IBM WebSphere 3.5 (EJB 1.0) as the target application server.

When verifying and correcting code, Together removes features of EJB 2.0 that not defined in EJB 1.0 specification and adds features of EJB 1.0 specification. Together automatically makes the following changes in the EJB implementation:

- Removes `EJB Exceptions`.

- Adds `RemoteException`.
- Changes the returned value of the `ejbCreate` method to `void` for CMP EJBs and removes the `return` statement in the method body.
- Changes `Collection` class to `Enumeration` in `findNullAccounts` and `findBigAccounts` methods.
- Changes transaction attributes.
- Removes the `abstract` modifier in CMP EJB implementation classes and in accessors and mutators for business properties representing database fields for a CMP EJB.

Together verification/correction does not change any EJB components that are defined in the later specification but not defined in the earlier one (such as message-driven beans, which are defined only in the EJB 2.0 specification).

After automatic correction, you should analyze your code and correct it manually as needed.

Moving from EJB 1.0 to EJB 1.1

You can use the same technique discussed in the previous section to understand how Together corrects code when moving from EJB 1.0 to EJB 1.1 specifications. Consider a simple sample from IBM WebSphere 3.5 (EJB 1.0 specification). Attempt the verification and correction by using BEA WebLogic 5.1.0 (EJB 1.1 specification) as the target application server. The most dramatic changes occur in entity EJBs.

Transitions among application servers

When you transition a project from one application server to another, expect to make changes in the following places:

- **Server-dependent deployment descriptors:** The J2EE Deployment Expert generates all deployment descriptors, including server-dependent ones. You can edit these descriptors only during the deployment process.
- **Code of the client connected with the new application server:** You must change the `getInitialContext()` method (`Context.INITIAL_CONTEXT_FACTORY`) in the code of the client, according to the target application server. You must also change the localhost in the `main()` method (for examples, see "Developing and deploying clients for various servers" on page 429).
- **EJB Java code, if the two application servers support different EJB specifications:** Changes in the code connected with the difference between EJB specifications are described in "Transitions among specifications" on page 426.

- **Properties of the targeted application server:** Different application servers have different properties and should be defined for deployment in the control panel of the selected application server. You should fill in the server parameters on the server-specific page of the J2EE Deployment Expert.

  If your application works with a database, before deployment you may need to edit the file `config.xml` to create a database pool and a data source.

Developing and deploying clients for various servers

The client code depends on the selected application server. For example, the simple client for the `HelloWorld` project for BEA WebLogic application server (`$TGH$/samples/java/ejb/J2ee/HelloWorld`) is implemented with the following code.

```
package client.weblogic;
import javax.naming.*;
import java.util.Properties;
import hello.*;

/**
public class HelloClient {
  public static void main(String[] argv) {

    try{

      Properties props = System.getProperties();
      props.put(Context.INITIAL_CONTEXT_FACTORY,
                "weblogic.jndi.WLInitialContextFactory");
      props.put(Context.PROVIDER_URL, "t3://localhost:7001");
      Context ctx = new InitialContext(props);
      HelloHome home = (HelloHome)
          javax.rmi.PortableRemoteObject.narrow(
                ctx.lookup("hello.HelloHome"),HelloHome.class);
      Hello hello = home.create();
      System.out.println(hello.hello());
      hello.remove();
    }catch(Exception e){ e.printStackTrace(); }
  }
}
```

Together can generate a simple JSP client automatically. If you are going to redeploy your client to another application server, you should edit the client class and correct the code described below:

- *findByPrimaryKey* method**:** If your project contains CMP Entity EJBs, it uses the `findByPrimaryKey` method. The signature of the `findByPrimaryKey` method varies among different EJB specifications.

  With the EJB 2.0 specification, you should use

  ```
  findByPrimaryKey(id).
  ```

In the EJB 1.0 specification you should use

```
findByPrimaryKey(new <Name>PK(id))
```

where `<Name>PK` is the returned type of `Primary Key`.

- localhost: You should correct the `main()` method according to the application server. For example, for BEA WebLogic 6.0, use:

  ```
  String url ="t3://localhost:7001"
  ```

  For IBM WebSphere 3.5, use:

  ```
  String url ="iiop://localhost:900"
  ```

- *getInitialContext* method: The `getInitialContext()` method varies among application servers. For example, for BEA WebLogic 6.0 you should write:

  ```
  h.put(Context.INITIAL_CONTEXT_FACTORY,
  "weblogic.jndi.initialContextFactory")
  ```

  For IBM WebSphere 3.5 you should write:

  ```
  h.put(Context.INITIAL_CONTEXT_FACTORY,
  "com.ibm.ejs.ns.jndi.CNInitialContextFactory")
  ```

- lookup: In CMP EJBs, the lookup method varies among application servers. For example, for BEA WebLogic 6.0, you should write this in lookup home:

  ```
  Object home = (<Name>Home) ctx.lookup("containerManaged.AccountHome")
  ```

  For IBM WebSphere 3.5, you should write:

  ```
  Object home = (<Name>Home) ctx.lookup("<Name>Home")
  ```

Encoding in deployment descriptors

If you want to retain your encoding in the rest of your project, you can “turn off" encoding for deployment by using a special option that does not add the encoding information to the deployment descriptor. In this case the target application server will consider the text of the deployment descriptor as written with standard UTF-8 or UTF-16 encoding. One result is that you will not be able to use any special symbols or characters in the deployment descriptor.

To use the standard encoding for deployment descriptors:

1. Open the Options dialog from the main menu by choosing **Tools | Options | <level>**.
2. Click *EJB* on the left pane.
3. Check **Do not write encoding declaration for deployment descriptors** on the right pane.

Chapter

28

J2EE Platform Security Support

The security features of an enterprise or Web application identify which users or other entities can access the application or components of the application. Security features are implemented in deployment descriptors and in code.

This chapter discusses the support in Together for J2EE security features. It includes the following topics:

- "Security support in J2EE diagrams" on page 431.
- "Security in EJB modules" on page 433.
- "Security in Web applications" on page 437.
- "Security in enterprise applications" on page 440.
- "Security in resource adapters" on page 440

Security support in J2EE diagrams

J2EE security features are intended to provide security for each tier of components of enterprise and Web applications. Primary J2EE security features include security roles, security role references, principals, security constraints, security and method permissions, Web resource collections, and authentication mechanisms.

Together enables you to visually model security features for your application through special security elements on J2EE diagrams, EJB security role references, and links defining the relationships among security elements and application components.

Together translates some of the security elements on diagrams directly into code. The J2EE Deployment Expert captures information for security design elements in deployment descriptors when you deploy from Together.

Security elements in J2EE diagrams

J2EE security elements on J2EE diagram vary with the type of diagram. Table 47 lists all of the security elements along with their corresponding diagram types.

Table 47 J2EE security elements on diagrams

Button	Meaning	Diagram Types
	Security Role: A category of application users. Corresponds to `<security-role>` in a deployment descriptor.	EJB assembler, Web application, enterprise application
	Principal: A user or a group of users who can be authenticated.	EJB assembler, Web application
	Method Permission: Indicates which roles are allowed to access which methods. Corresponds to `<method-permission>` in a deployment descriptor.	EJB assembler
	Web resource collection: A a set of URL patterns or HTTP methods that describe resources to be protected. Corresponds to `<web-resource-collection>` in a deployment descriptor.	Web application
	Security constraint: Determines who can access a Web resource collection. This includes authorization constraints and user data constraints. Corresponds to `<security-constraint>` in a deployment descriptor.	Web application
	Link: Provides mapping between security elements.	EJB assembler, Web application
	Authentication Mechanism: Describes how users of resources are to be authenticated. Corresponds to `<authentication-mechanism>` in a deployment descriptor.	resource adapter
	Security Permission: Indicates which permissions are allowed from specified code sources. Corresponds to `<security-permission>` in a deployment descriptor.	resource adapter

Security roles

A *security role* is a category of users which is defined by the role they play with respect to an enterprise or Web application. Security roles are the central J2EE feature that supports authorization. Together provides a security role elements on EJB assembler, enterprise application, and Web application diagrams.

Mappings between security roles and other J2EE security elements indicate who or what entity can access all or specific parts of an application. These elements include:

- **Principals:** users (in a security policy domain) or groups (in an operational environment) that can be authenticated. There are special diagram elements for principals.
- **Method permissions:** indicate which security roles can access a collection of methods. There are special diagram elements for method permissions.
- **Security role references:** EJB attributes that define security roles.

Entity and session EJBs on Together diagrams have right-click menu commands for inserting security role references into the bean code. Security role references display in a bottom compartment of an EJB in a light blue color.

Linking security elements

Together provides a link diagram element on EJB assembler and Web application diagrams to establish mappings between different security elements. You can link the following types of elements and features:

- Security roles to principals
- Method permissions to security roles
- Methods to method permissions
- Security role resources to security roles
- Security constraints to Web resource collections
- Security constraints to security roles

When you link EJB methods to a method permission, you establish the access permission for those methods. When you link that method permission to a security role, you establish the roles that users must be playing to access the methods. When you link that security role to principals, you establish who can access the methods.

Some of the linking is optional. However, there are two restrictions on linking security roles and security role elements:

- If an EJB defines a security role reference (shown in the last compartment of the EJB node in the diagram), the reference must be linked to exactly one security role element.
- One security role element can be linked to any number of EJB security role references or method permissions, including none at all.

Security in EJB modules

There are two variants of security methodologies in EJB modules:

- **Declarative**: The security features of the application are defined external to the Java code, usually in a deployment descriptor.
- **Programmatic:** The security features of the application are defined within the application code.

Together supports both variants in EJB assembler diagrams. Recall that an EJB assembler diagram represents a single EJB module.

Declarative security and method permissions

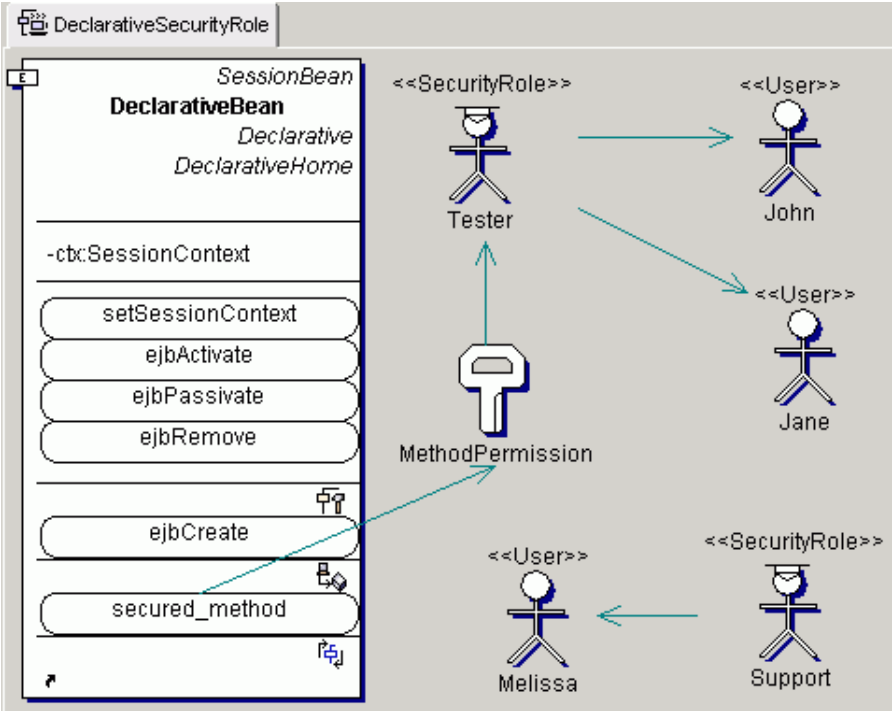
In declarative security, all of the security is defined external to the application. No special security code is required in the EJB. The `<method-permission>` element of the deployment descriptor determines the list of methods that can be accessed by users with a certain security role. On an EJB assembler diagram, the method permission diagram element is critical for declarative security.

To implement declarative security in an EJB module:

1. Open the EJB assembler diagram representing your EJB module.
2. Determine which methods you want to be secured in the EJBs on the diagram. In Figure 110, `secured_method` is such a method.
3. Place a method permission element on the diagram ( ) and link the secured method to the method permission.
4. Place security role elements on the diagram ( ), editing their names as desired. These names are used in the deployment descriptor. Figure 110 shows an EJB assembler diagram with two security roles, `Tester` and `Support`.
5. Place principal elements on the diagram ( ) and set their properties.
    1. To set the properties, right click the principal and choose **Properties**. This opens the inspector.
    2. Open the **Properties** tab. of the inspector. You can change the name; you can set the element stereotype to *User* or *Group.*

    Figure 110 has three principals, `John`, `Jane`, and `Melissa`. All of them are users.

Figure 110 EJB assembler diagram showing a declarative security role



6. Link each security role to the principals making up that role.
7. Link the method permission to a security role.

   Figure 110, shows that the only users who can access `secured_method` are `John` and `Jane`.

It is not necessary to link each security role to a method permission. For example, Figure 110 shows `Support`, which is not linked to any method permission. In that case, the J2EE Deployment Expert adds the `Support` security role and its user `Melissa` to the deployment descriptor.

Programmatic security and security role references

In programmatic security, the security features of an application are built into the Java code. Two methods implement programmatic security for EJB modules in conjunction with security roles:

- `isCallerInRole`: tests if the caller has a given security role with regard to the security context of the client
- `getCallerPrincipal`: obtains a `java.security.Principal` interface for the current caller.

`isCallerInRole` requires a `<security-role-ref>` element in the deployment descriptor for all the security role names used in the EJB code. The J2EE Deployment Expert creates such deployment descriptor elements by making use of security role references, which are attributes of the EJB code.

To create a security role reference in an EJB module:

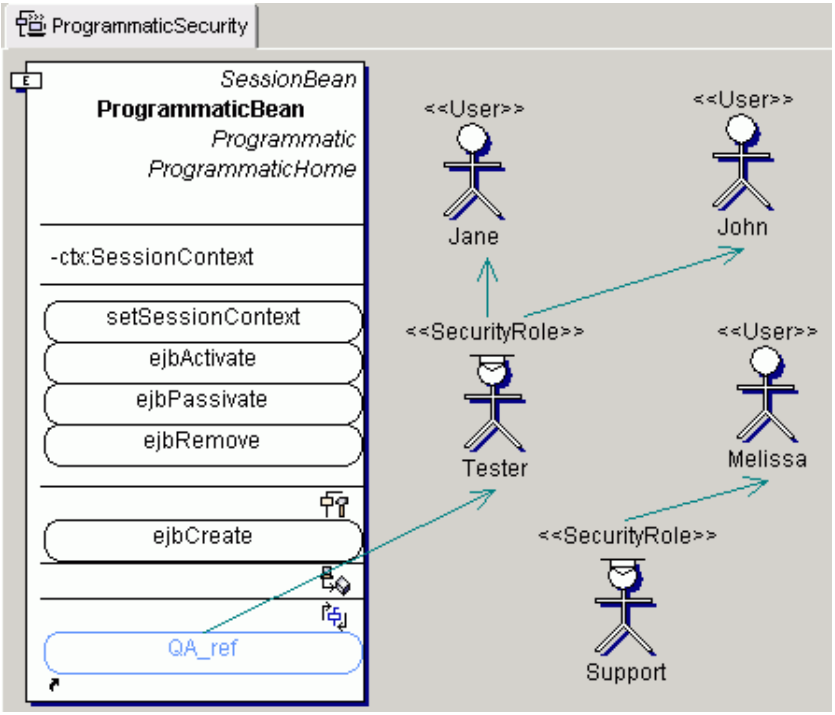
1. Open the EJB assembler diagram representing your EJB module.
2. Create security role references in the EJBs as required.
    1. Right click the EJB and choose **New | EJB Security Role Reference**.
    2. Edit the name when you create the reference.

    This defines an attribute in the bean representing the security role reference. Figure 111 shows an EJB assembler diagram with an EJB named `ProgrammaticBean` and its security role reference named `QA_ref`.
3. Place security role and principal elements on the diagram as required. See "Declarative security and method permissions" on page 434 for details.
4. Link the EJB security role reference to the corresponding security role elements. Figure 111 shows the link from `QA_ref` with the security role `Tester`. In this example, the access of `Jane` and `John` to the `ProgrammaticBean` is defined by the `Tester` security role.

You can also place a security role element on the diagram and link it to some principal element without any connection to a concrete method. Figure 111 shows this variant where the `Support` security role links to the `Melissa` principal.

If an EJB has a security role reference linked to a security role, then the J2EE Deployment Expert will generate information about properties of the reference and its associated security role in the deployment descriptor.

Figure 111 Programmatic Security Role



Security in Web applications

Three elements on Web application diagrams support the security of Web applications: security roles, security constraints, and web resource collection.

Note You can specify the user authentication options on the Login Config tab of the Web Properties tab in the diagram inspector. For details, see "Login Config tab" on page 364.

Security roles and principals

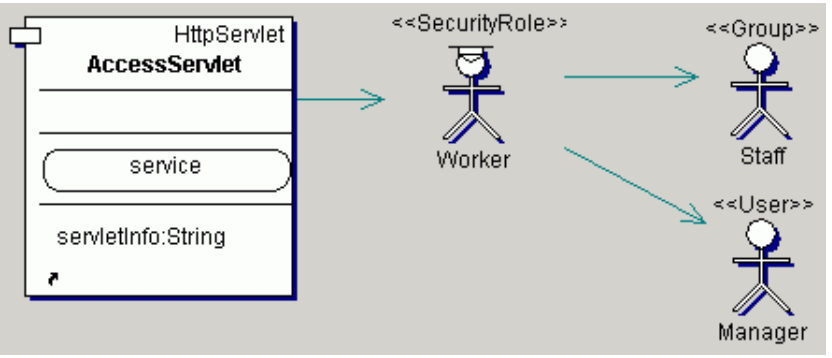
The Web container determines which users have access to which parts of a Web application. The deployment descriptor created by the J2EE Deployment Expert can provide that kind of information from security roles on Web application diagrams.

To use a security role on a Web application diagram:

1. Open the Web application diagram corresponding to your application. The diagram should contains the elements necessary for your project (shortcuts of servlets, JSPs, Web files, and so on). Figure 112 shows a Web application diagram with a single code element, which is a shortcut to a servlet named `AccessServlet`.
2. Place a security role element on the diagram ( ). Set the name for this security role. This name will be used in the deployment descriptor. Figure 112 shows a single security role named `Worker`.
3. Place principal elements on the diagram ( ) and set their properties. Figure 112 shows two principals, `Staff` and `Manager`. The stereotype for `Staff` is `Group`, while the stereotype for `Manager` is `User`.
4. Link the shortcut to the security role.
5. Link the security role to its principal users.

In Figure 112, the access of `Staff` and `Manager` to `AccessServlet` is defined by `Worker`.

Figure 112 Using a security role design element for supporting security



Constraints and Web resource collections

A Web resource collection is a set of URL patterns and HTTP methods that describe Web resources to be protected, such as images and JSPs. You can specify the protection in Web application diagrams by using a security constraint and security roles. The security constraint determines who is authorized to access the Web resource collection.

To specify security constraints for a Web resource collection:

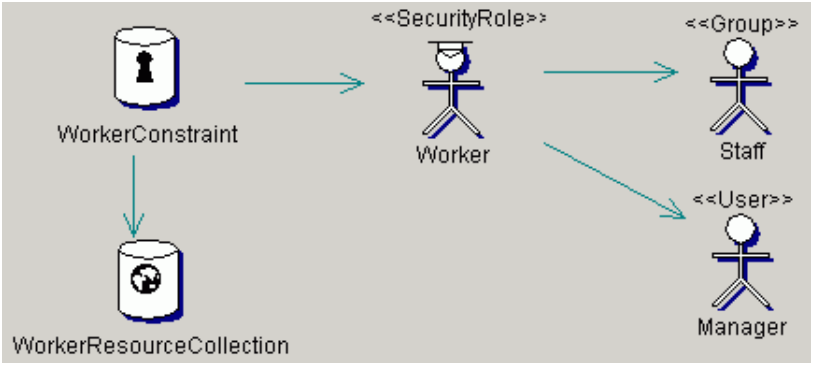
1. Open the Web application diagram corresponding to your Web application.
2. Place a Web resources collection element on the diagram ( ) and set its properties.
    1. Right click the Web resources collection element and choose **Properties**. This opens the inspector.

2. Open the **Web resource collection properties** tab.
3. Fill the **URL pattern** field with the path to the protected resource files. For multiple patterns, click the button on the right of the field.
4. Fill the **HTTP method** field with the appropriate HTTP methods for the resources (such as `GET` and `POST`). For multiple methods, click the button on the right of the field.

3. Place a security constraint element on the diagram ( ) and set its properties.
   1. Right click the security constraint element and choose **Properties**. This opens the inspector.
   2. Open the **Security constraint properties** tab.
   3. In the **User data constraints** field, choose one of the following transport guarantees:
      - **NONE**
      - **INTEGRAL**: data cannot change while transmitting between client and server
      - **CONFIDENTIAL**: no other entities can observe the contents of the transmission
4. Draw a link from the security constraint to the Web resources collection.
5. Create security roles and principals as described in "Declarative security and method permissions" on page 434.
6. Draw a link from the security constraint to the appropriate security role.

In Figure 113, the Web container grants users access to the `WorkerResourceCollection` of Web resources only after it determines that they are in a `Worker` role, which means that they are `Manager` or members of the `Staff` group.

Figure 113 Links between security supported elements



Security in enterprise applications

A security role element on an enterprise application diagram defines the security properties for all elements on the enterprise application diagram. When you deploy from the enterprise application diagram, the J2EE Deployment Expert generates the lines corresponding to a security role in the deployment descriptor

Security in resource adapters

Two diagram elements that support security in resource adapter diagrams: authentication mechanisms and security permissions.

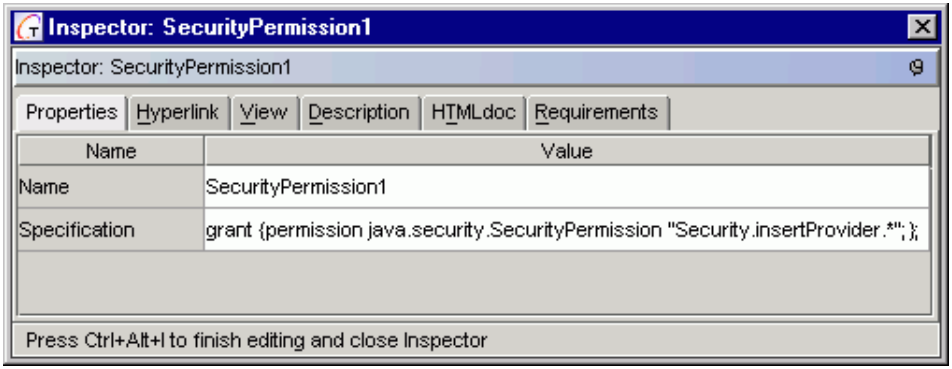
The security permission element corresponds to the deployment descriptor tag `<security-permission>`, which in turn contains a `<security-permission-spec>` tag. These tags describe the permission to be granted to each resource adapter.

To describe a permission for a resource adapter:

1. Open the resource adapter diagram corresponding to the RAR file.
2. Put a security permission element on the diagram ( ).
3. To set the properties of the security permission element, right click the element and choose **Properties**. This opens the inspector.
4. Open the **Properties** tab of the inspector. Fill in the **Specification** field with the appropriate grant statement, such as the following one shown in Figure 114:

```
grant {permission java.security.SecurityPermission "Security.insertProvider.*";};
```

Figure 114 Setting security permissions



The authentication mechanism element corresponds to the deployment descriptor tag `<authentication-mechanism>`. It describes how the resource adapter user is to be authenticated.

To specify authentication mechanisms for RAR files:

1. Open the resource adapter diagram corresponding to the RAR file.
2. Put an authentication mechanism element on the diagram ( ).
3. To set the properties of the authentication mechanism element, right click the element and choose **Properties**. This opens the inspector.
4. Open the **Properties** tab of the inspector. Choose settings for the following:
   - **Authentication mechanism**: Choose **BasicPassword** (Basic user-password- based authentication specific to an EIS) or **Kerbv5** (Kerberos version 5-based). This field corresponds to the deployment descriptor tag `<authentication-mechanism-type>`.
   - **Credential interface**: Choose **javax.resource.spi.security.PasswordCredential** or **javax.resource.spi.security.GenericCredential**. This field corresponds to the deployment descriptor tag `<credential-interface>`.

Part

7

EXTENDING TOGETHER

•Chapter 29, “Advanced Customizations”

•Chapter 30, “Together Open API”

Chapter

29

Advanced Customizations

This chapter is intended for administrators or managers who need to create shared custom configurations or modify Together in order to meet corporate standards. Advanced users interested in the low-level customization capabilities of Together may also be interested in this information. Note that the most commonly-needed customizations of the configuration properties can be completed from the Options dialog. To invoke the dialog, choose Options from the Tools menu.

If you are interested in more detailed information beyond what is provided in this chapter, you can review the underlying configuration files located in the `./config` directory of the installation. Customizations that you can not configure by setting options may be possible by programming the Together API.

This chapter includes the following topics:

- "Customizing the user interface" on page 445.
- "Creating custom diagram types" on page 446.
- "Customizing Property Inspectors" on page 453.
- "Customizing View Management Show options" on page 458.

Customizing the user interface

Customizing the user interface requires editing the underlying configuration properties files located in the `./config` directory of the Together installation. Configuration files have the `.config` extension.

It may also be necessary to edit the resource files referenced by lines in the configuration files. These are located in the `./lib/i18n` directory of the Together installation and have the `.properties` extension. The files in this directory are of particular interest if you want to localize the Together installation, as they contain UI strings that would need to be translated to another language.

Important Before modifying any of the configuration or resource properties files, create a backup copy of the original files.

Creating custom diagram types

Together supports the currently-defined UML diagram types, and Business Process, and special Together diagrams, but it does not confine you to using only these types. Together technology lets you define your own custom diagram types.

To create custom diagram types, complete the following steps:

1. Create diagram icons and icon references.
   - Create icon images to display in the Together user interface.
   - Create a folder to store the custom diagram icons and add it to the paths in the `Together.bat` file.
   - Edit the `$TGH$\config\resource.config` file to provide references to the icons of the created diagram type.
2. Create a diagram configuration file.
   - Create a new configuration file in `$TGH$\config` for each custom diagram type.
3. Define tree-view icons.
   - In the configuration file, define the diagram entity and name, and create references to the icons that will be displayed in the Explorer tree-view.
4. Define toolbar icons.
   - In the configuration file, define the diagram entity and name, and create references to the toolbar buttons.
5. Define the diagram elements for the new diagram.

The procedure is described in the following sections. You can also refer to `$TGH$/doc/guides/config/index.html`.

Step 1: Creating icons and icon references for the UI

Two icons are required for each diagram type to be represented in the Together UI: a small one for the Explorer tree-view, and a large one for the New Diagram dialog. The icon images should meet the requirements specified in Table 48.

Table 48 Specifications for graphical icon images

Icon	Pixel Dimensions	File Format
small (tree-view)	16 x 16	GIF (transparent bg)
large (dialog)	32 x 32	GIF (transparent bg)

Editing the Together.bat file

Add the path to the folder where the icons are stored to the `-cp` variable of the `Together.bat` file.

Example:

```
-cp "$TGH$\images;..."
```

Referencing images in the config file

To refer to the icon images of the new diagram, add the following lines to the `resource.config` file:

```
resource.element.{diagram_unique_name}.name = "{diagram_name_1}"
resource.element.{diagram_unique_name}.diagramName = "{diagram_name_2}"
resource.element.{diagram_unique_name}.icon.small = "{path_to_the_icon}"
resource.element.{diagram_unique_name}.icon.large = "{path_to_the_icon}"
```

where:

- `diagram_name_1` is the name used for the new diagram.
- `diagram_name_2` is the name used in the New Diagram dialog box.
- `path_to_the_icon` is the path relative to the folder specified in the -cp variable of Together.bat.
- `resource.element` is a predefined construction to denote resources: icons, names, buttons.

Example:

```
resource.element.Heffalump_Trap_Diagram.name = "Heffalump Trap Diagram"
resource.element.Heffalump_Trap_Diagram.diagramName = "Heffalump Trap
Diagram"
resource.element.Heffalump_Trap_Diagram.icon.small = "small_icon.gif"
resource.element.Heffalump_Trap_Diagram.icon.large = "big_icon.gif"
```

Step 2: Creating the diagram configuration file

1. Create a new file `$TGH$\config\{diagram_name}.config`.

2. Enter at least one line, which defines a new diagram type:

```
model.diagramType.{diagram_unique_ID}={diagram_unique_name}
```

where:

- `diagram_unique_ID` is any word or number that will be used to sort diagram icons in the New Diagram dialog box. If a number is used, it should be greater than 9 because numbers from 1 to 9 are reserved for standard UML diagrams.
- `diagram_unique_name` is any word that will be used to refer to this new type of diagram.
- `model.diagramType` is a predefined construction to define the new diagram type.

Example:

```
model.diagramType.htd=Heffalump_Trap_Diagram
```

Step 3: Defining tree-view icons

First, define icons for the new diagram elements displayed in the Model tree-view.

To define the tree-view icons:

In the diagram configuration file, add a line for each newly created diagram type:
`resource.element.{element_unique_name}.icon.small = "{path_to_icon}"`
where:
`element_unique_name` is a word denoting the new element type.

Note `element_unique_name` may not contain spaces. Use underscores instead of spaces.

Example:

```
resource.element.sdEntity.icon.small = "Images/tv-deepPit.gif"
resource.element.sdAttribute.icon.small = "Images/tv-honey.gif"
```

Step 4: Defining toolbar icons

Button definitions dynamically construct Toolbar icons. Each toolbar button represents one element, and every element in a diagram is handled as either a node or a link. Any element drawn as a type of line or arrow between nodes defines a link. In some situations, there are important differences between nodes and links. However, in this task, the name variable to store the new element represents the only difference between handling buttons for a node or a link. Accordingly, the name of the variable to store the new element must be either `node` or `link`, and the internal function used to insert the element into the new diagram must be either `createNode` or `createLink`.

The example that follows demonstrates adding a node element to the diagram. The definition of each button has the following format:

```
diagram.toolbar.button.{diagram_unique_name}.{element_toolbar_name} = node
= createNode("{element_unique_name}")
```

```
diagram.toolbar.button.{diagram_unique_name}.{element_toolbar_name}.condit
ion = "{enabling_condition}"
```

```
diagram.toolbar.button.{diagram_unique_name}.{element_toolbar_name}.icon =
"{path_to_icon}"
```

where:

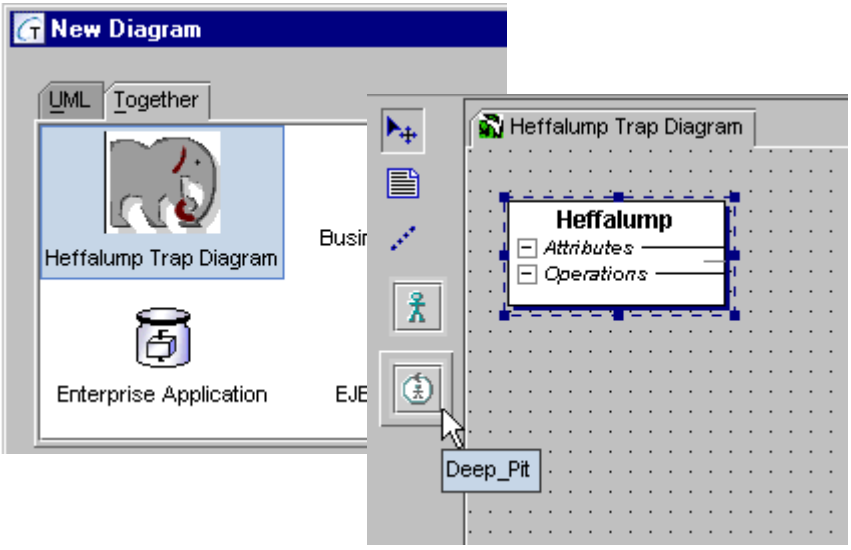
- `element_toolbar_name` is a unique name of a toolbar button.
- `path_to_icon` is the path to the button's icon.
- `node` is a variable that stores the new element.
- `createNode` is an internal function that creates a new node (entity).
- `enabling_condition` is any valid boolean expression that defines the availability of the button.
- `diagram.toolbar.button` is a predefined construction to define a new button on the toolbar.

Tip To create a link element, change `node` to `link`, and change `createNode` to `createLink`.

Example:

- `diagram.toolbar.button.Heffalump_Trap_Diagram.Deep_Pit = node = createNode("Deep Pit")`
- `diagram.toolbar.button.Heffalump_Trap_Diagram.Deep_Pit = %DEFAULT_LANGUAGE% != "idl"`
- `diagram.toolbar.button.Heffalump_Trap_Diagram.Deep_Pit.icon = "toolbar_icon_2.gif"`

Figure 115 Custom diagram and toolbar buttons



Step 5: Defining viewmaps

In order to create new elements in a diagram, it is necessary to describe the graphical presentation of each one in the `.config` files for the new diagram type. A construction similar to the one shown below must be added to the `.config` file of a new diagram type. This section presents each step based on the sample `.config` file.

You can refer to `$TGH$/doc/guides/config/Viewmaps.html` for detailed information on viewmaps.

Defining graphical presentation

Define graphical presentation starting with the header

```
view.map.*.{element_unique_name}.isTopLevel() =
```

where:

`view.map.*` is a predefined construction.

`element_unique_name` is the element name defined in the previous section.

`isTopLevel()` means that all following operations are performed if the element is on the diagram top level.

Example:

```
view.map.*.sdEntity.isTopLevel() =
```

Defining the shape of an element

The diagram manages the nodes in terms of the boundaries each node fits inside of, not the drawn shape inside of those boundaries.

Define the shape of the new element:

```
\ setGraphicObject("{shape_name}");
\ setLayoutConstraints( {parameter}, [parameter], ... );
```

where:

- `setGraphicObject` is a function that sets the shape_name of an element. The predefined shapes are Cube, RectangleVisible, Circle, RoundRectangle, Oval, Folder, $Note, Actor, BeanComponent, RectangleObject, $RoundRectangle, $GraphSequenceInteraction, RectangleObjectBordered, MultipleInstance, Component, Folder, Oval, DeepHistory, History, ArcRectangle, Rhomb, ConcavePentagon, and ConvexPentagon.

Tip To see examples of the shape_names and how they are used, see `$TGH$\config\view.config` file.

- `setLayoutConstraints` sets parameters of the element presentation. This function is optional.
- `parameter` is name-described. See the example below.

Example:

```
\ setGraphicObject("Cube");
\ setLayoutConstraints(minWidth(20),minHeight(20),
\              preferredWidth(100),preferredHeight(100),
\              canShiftX(true),canShiftY(true));
```

The above example draws the node inside of a cubical boundary.

Defining the element's options

This section defines options for the element's graphical presentation.

Enable display of the element's unique name

```
\ name = addCompartment("RectangleInvisible","Name");
\ name->setLayoutConstraints(horizontalAlign("left"),
   \ verticalAlign("top"),widthAlign("parentDefined"));
```

Here register a new compartment with the name `Name` and shape `RectangleInvisible` and set its layout.

Add the element's name to the compartment

```
\  nameLabel = addToCompartment(label(getProperty("$name")), "Name");
```

Here the function `addToCompartment` adds a new label to the compartment Name. The function `getProperty` returns a value of `$name` for this element, which was defined in the beginning of the config file as the `uniqueName` for the element.

Enable in-place editing

This line enables in-place editing of the element's name in the label. The expression `property:= "$name"` shows the element property to be updated.

```
\ nameLabel-> setInplaceEditor({property:="$name",default:=true});
```

Define label properties

The following lines set label properties:

```
\ nameLabel->setAlignment("Center");
\ nameLabel->setLayoutConstraints(preferredHeight
(16),fixedHeight(true));
```

Define links

To enable element links to other elements, add the following line:

```
\ setCanHaveLinks()
```

Tip To see other functions and options that are available for config files, see `$TGH$\doc\guides\config\index.html`. Choose *Viewmap queries*.

See the complete sample config file below.

Sample configuration file

```
############################################################
# Defining the new diagram type: shape type, name, icons
#
model.diagramType.sd=SampleDiagram
resource.element.SampleDiagram.name = "Sample Diagram"
resource.element.SampleDiagram.diagramName = "Sample"
resource.element.SampleDiagram.icon.small = "Treeviews/tv-sam-diagram.gif"
resource.element.SampleDiagram.icon.large = "DiagramTypes/
SampleDiagram.gif"
############################################################
# Defining icons for the new elements. These icons will be
# used in treeview
#
resource.element.sdEntity.icon.small = "Treeviews/tv-sdEntity.gif"
resource.element.sdAttribute.icon.small = "Treeviews/tv-attrib.gif"

############################################################
# Defining toolbar for the diagram
#
diagram.toolbar.button.SampleDiagram.Entity =
\ node = createNode("sdEntity");
```

```
\ node->setProperty("uniqueName","Entity")
diagram.toolbar.button.SampleDiagram.Entity.icon = "SampleDiagram/
entity.gif"
diagram.toolbar.button.SampleDiagram.Relationship =
\ link = createLink("sdRelationship");
\ link->setProperty("uniqueName","Relationship")
diagram.toolbar.button.SampleDiagram.Relationship.icon = "SampleDiagram/
relationship.gif"


############################################################
# Defining viewmap for the diagram
#
view.map.*.sdEntity.isTopLevel() =
\ setGraphicObject("Cube");
\ setLayoutConstraints(minWidth(20),minHeight(20),
\ preferredWidth(100),preferredHeight(100),
\ canShiftX(true),canShiftY(true));
\ name = addCompartment("RectangleInvisible","Name");
\ name->setLayoutConstraints(horizontalAlign("left"),
\ verticalAlign("top"),widthAlign("parentDefined"));
\ nameLabel = addToCompartment(label(getProperty("$name")),"Name");
\ nameLabel->setInplaceEditor({property:="$name",default:=true});
\ nameLabel->setAlignment("Center");
\ nameLabel->setLayoutConstraints(preferredHeight(16),
\ fixedHeight(true));
\ setCanHaveLinks()
```

Customizing Property Inspectors

The abstract model represents properties of an object in a structured way. The Inspector content varies depending on the IDE context that contains information about the selected element. For example, if you look at the Inspector for a regular Java class and an EJB implementation class, the content displayed is quite different. The Inspector consists of several components, each representing a group of properties, their names, and values. Property Inspectors are flexible and can be customized. You can create custom tab pages, add new fields, change field names, add new stereotypes, and make other changes.

There are multiple ways to customize the Inspector, including:

- Using the Together Open API
- Using the Custom Properties module (provides convenient visual customization)
- Enabling the config-based inspector

Adding custom pages and fields to the Inspector

API-based Inspector customization

If you want to create your own page or add new properties to an existing Inspector, you can use the Inspector API. Inspector is a startup module located in `$TGH$\modules\com\togethersoft\modules\`, and it does not display on the Modules tab of the Together Explorer. If you need to customize the Inspector, you should edit the appropriate manifest files and classes. The updated startup module activates upon restart of Together.

Tip API-based customizations are developed using Java. If you are a C++ developer, and unaccustomed to developing in Java, there are tutorials located at the following: `http://java.sun.com/docs/books/tutorial`

Classes that enable adding new pages and fields to the Inspector are not included in the Inspector module, but reside in `$TGH$\modules\com\togethersoft\modules\inspector\examples`.

Note Before proceeding with the steps below, create a backup copy of the class `AddPageToInspector.java` (or `AddFieldsToInspector.java`) located in the `$TGH$\modules\com\togethersoft\modules\inspector\examples` directory.

To add new pages to the property inspector:

1. Using a text editor, Open the appropriate manifest file (`*.def`) in `$TGH$\modules\com\togethersoft\modules\inspector\examples`. For this example, open `Page.def`. (To modify fields, open `Field.def`).
2. In the `Page.def` file, uncomment these lines:

```
MainClassName =
      com.togethersoft.modules.inspector.examples.AddPageToInspector
Time = Startup
```

3. Close `Page.def` saving the changes.
4. Open JBuilder, and create a new project. Choose **File | New Project** from the main menu. The New Project Wizard opens.
5. Enter a project name, for example: `Inspector`.
6. For the project directory, click the select button, and browse to the `$TGH$\modules\com\togethersoft\modules\inspector\examples` directory. Click **Next**.
7. To specify the source path for the project, click the *Source* tab, select the source path, and click **Edit**. Browse to the `$TGH$\modules\com\togethersoft\modules\inspector\examples` directory and click **OK**.
8. Select the *Required Libraries* tab, and add the following archive files:
   - `$TGH$\lib\openapi.jar`
   - `$TGH$\lib\together.jar`
   - `$TGH$\modules\com\togethersoft\modules\inspector\inspector.jar`

9. Click **Finish** to create the project.

10. From the JBuilder Project Pane, open `AddPageToInspector.java`, and edit the source code, specifying the desired page and field names as shown in the sample code below:

```
...
IdeInspectorProperty property;


// Replace "myStringProperty" with the custom page name
// This creates a string property
property = new RwiInspectorStringProperty(rwiElements, " taskDescription
");


// Specify custom name to be shown in the UI
// Replace “My string property” with the name to be shown in the UI
property.setName(" Task manager ");


// adds it to the page
// default editor - just a text field - is used
page.addProperty(property, null);
// Replace "myBooleanProperty" with the custom name
property = new RwiInspectorBooleanProperty(rwiElements, " isReady ");


// Specify custom name to be shown in the UI
// Change "My boolean property" to the name to be shown in the UI
property.setName(" Is the task completed ");


// adds it to the page
// shown as a checkbox in the UI
page.addProperty(property, null);


// Creates a string property
// Replace "myChoiceProperty" with the custom string property name
property = new RwiInspectorStringProperty(rwiElements, " rate ");
```

```
// Specify custom name to be shown in the UI
// Replace "My choice property" with the name to be shown in the UI
property.setName(" Select rate from the list ");

// Creates and sets an editor for it - combobox(choice)
// Specify custom names for the drop-down list
// Replace "value1", "value2", "value3" with "rate1", "rate2", "rate3"
SwingComboBoxEditor editor = new SwingComboBoxEditor(
  new DefaultComboBoxModel(new String[] {"rate1", "rate2", "rate3"}));
...
```

Compiling

1. Compile `AddPageToInspector.java` using the javac compiler. From the JBuilder main menu, choose **Project | Project Properties**.
2. Select the *Build* tab. On the Build page, select the *Java* tab.
3. Click the drop down arrow for the *Compiler* field, and choose javac from the list. Click **OK**.
4. From the JBuilder main menu, choose **Project | Make "AddPageToInspector.java"** to compile the changes.

Showing new pages and fields in Together:

1. Place the compiled class files in the `$TGH$\modules\com\togethersoft\modules\inspector\examples` directory.
2. Restart Together if it is already running.
3. Open a project, and select a class element.
4. Right-click on the class, and select **Properties**. The tab for the new page defined in `AddPageToInspector.java` displays in the Inspector.

More API documentation

The best way to learn how to write an inspector is to study the source code of the Inspectors delivered with Together. The open API provides some technical documentation of the key classes and interfaces, and a commented example of how to perform a simple Inspector customization. Refer to the documentation for `com.togethersoft.openapi.ide.inspector` through `$TGH$/doc/api/index.html`.

Customizing Inspector by means of the Custom Properties Module

It is possible to visually customize the Inspector for a particular object. This is done using the Custom Properties module. Each new property added to the selected object adds an appropriate tag to the source code. However, these changes apply to the object in reference only. Other objects of the same type are not affected.

To add or delete custom properties:

1. Make sure that the module Custom Properties is activated.
    1. Choose **Tools | Activate/Deactivate Features** from the main menu.
    2. Go to the Together Features tab.
    3. Check the *Custom Properties* box, if it is not checked.
2. Invoke the Inspector for the selected object. The Custom Properties tab appears in the Inspector.
3. On the Custom Properties tab, click the **Add** button.
4. Enter the property's name and value in the added field. Add as many properties as required.
5. Click **Remove** to delete the selected properties.
6. Press **`Ctrl+Enter`** to apply the changes and close the Inspector.

User-defined inspector customization with the Inspector Property Builder

In the Options dialog, the General options group includes the *Support user-defined inspector* checkbox, which controls whether inspector pages are built from config entries, and enables visual creation of additional user-defined pages. When this option is checked, the Tools menu displays the Customize Inspector command, which opens the Inspector Property Builder dialog.

A new page with the specified property name appears on the Inspector for the selected elements. Together writes relevant entries to the `$TGH$/config/changes.config` file.

To customize the inspector using the Inspector Property Builder:

1. In the Options dialog, make sure that the option *Support user-defined inspector* is checked. From the main menu in Together, choose **Tools | Options | Default Level** or **Project Level**. Choose **General** on the left of the dialog, and check **Support user-defined inspector** on the right of the dialog.
2. In the diagram, select the component for which the properties inspector should be customized.

Tip When working with the Inspector Property Builder, keep in mind that:

- The dialog is non-modal. This means that you can navigate over the diagram, and every time you click on another component, the new selection displays in the Choose shapetypes field.

- You can select multiple components. Holding down the Ctrl key, click on the desired components. Their names are added to the Choose shapetypes field.

3. Choose **Tools | Customize Inspector** on the main menu.
4. In the Inspector Property Builder dialog, specify the property name, visible name, component, and type of the field.
5. Click **Add** to create a new entry. The changes apply the next time you click the component on the diagram.
6. Use the **Delete** button to remove any unnecessary entries.

For a detailed description of the controls, refer to the Inspector Property Builder section of the Online Help.

Customizing View Management *Show* options

The Options dialog provides the Show group of options under the View Management node to control the presentation of information in diagrams. These options result in configuration properties contained in the `filters.config` file and determine which elements are hidden in diagrams. By default, nothing is hidden.

At the properties level, Together essentially defines filters that remove the defined elements from view when the option is activated (hence the filename `filters.config`). The default state of the filters is `off`, meaning that the defined elements are not filtered (that is, they are shown). At the UI level, when a filter in the properties file is off, then the option's value ("show") is true and the option is checked in the Options dialog.

All of the Show options can be user defined, in addition to the options labeled *User-defined*. If necessary, you can change the option text, remove options from the dialog, or create additional custom filters.

Changing the display text of a Show option

You might want to change the name of the options that appear in the Options dialog (for example, to accommodate international users). By default, the names of the options are extracted from a resource file. Consider the following line from the `filter.config` file:

```
optionsEditor.item.View Management.item.Filters.item.A.item.shortName.name
= ["filters/filter_shortName"]
```

The text in brackets is a reference to a string in a resource file. To change the name that displays in the Options dialog, you should either edit it in the resource file, or replace the reference in the configuration file with a string literal in double-quotes.

To edit the resource file:

1. Open the resource file `$TGH$/lib/i18n/filters.properties` in a text editor.

2. Search for the name of the option (for example, "All Packages"). The search should turn up a line similar to this: `all_packages=All Packages`.
3. Edit the name as desired.
4. Save and close the properties file.

The changes take effect the next time you start Together.

Tip Administrators with international users might want to make copies of the resource file for different languages and replace the English default file in the installations of non-English speaking users.

Removing a Show option from the Options dialog

If you do not want to use one of the predefined options and do not want it to appear in the Options dialog, comment out all lines of the option's section (for example, "All Classes") in the `filter.config` file. Update the sequential information (as described in the next section) to compensate for the removal of the commented section from the section sequence of the file.

Adding a Show option to the Options dialog

Conversely, you can create a new option and display it in the Options dialog by copying any of the existing sections and modifying the lines as necessary for the view you want. Note that the sections in the properties file are arranged sequentially and contain sequential information imbedded in the lines. When you add a section, you need to modify this information throughout the section so it is the last in the sequence.

For example, the section for the All Packages option is the first section and contains the following lines:

```
filter.a = hasProperty("$physicalPackage")
...
 optionsEditor.item.View Management.item.Filters.item.A.order = 10
```

The file has 12 sections. The above lines in the last section read as follows:

```
filter.l = hasProperty("$physicalPackage")
...
 optionsEditor.item.View Management.item.Filters.item.L.order = 120
```

If you copy the first section and paste it at the end of the file to create a 13th section, then you need to modify the lines as follows:

```
filter.m = hasProperty("$physicalPackage")
...
 optionsEditor.item.ViewManagement.item.Filters.item.M.order =  130
```

For all other lines in the section, you need to replace occurrences of **`.a`** with **`.m`** and occurrences of **`.A`** with **`.M`**. Then the sequential information in the section will be the highest in the alpha and numeric sequence.

Next, customize the filter definition in the first several lines of the section `(filter.[seq])`. The filter expression is contained in the first line of the section, such as:

```
filter.m = hasProperty("$physicalPackage")
```

The text in **bold** is the filter expression. This is usually a call to `hasProperty` or `hasPropertyValue`. Study the other filter expressions and observe their construction before coding your own filter expression.

For the remaining lines in this section, you can either use a reference to a resource:

```
filter.m.name = ["filters/my_new_filter_options"]
```

in which case you should add the property `my_new_filter` to the `filters.properties` file. Or you can use a string literal instead:

```
filter.m.name = "My New Filter Options"
```

Finally, you can update the other lines with references to resources or literals as required.

Chapter

30

Together Open API

Together is highly extensible through an open Java API. This chapter provides an introduction to the Together Open API and explains how to write your own modules.

This chapter consists of the following topics:

- "Overview of the Together Open API" on page 461.
- "Understanding modules" on page 462.
- "Extending Together using modules" on page 465.
- "Guidelines for developing modules" on page 467.
- "Tutorial: Writing and deploying modules" on page 470.
- "Accessing more API documentation" on page 480.

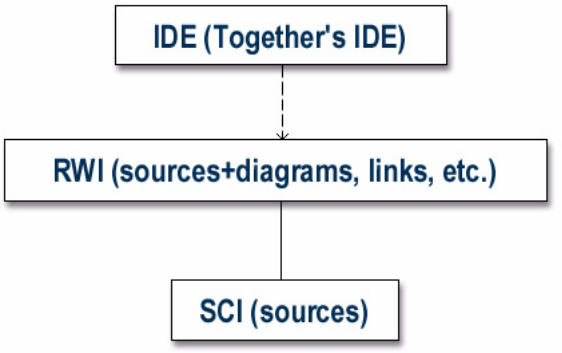
Overview of the Together Open API

The Together Open API consists of the Integrated Development Environment (IDE), Read-Write Interface (RWI), and Source Code Interface (SCI), packaged as follows:

- `com.togethersoft.openapi.ide` package and its subpackages
- `com.togethersoft.openapi.rwi` package and its subpackages
- `com.togethersoft.openapi.sci` package and its subpackages

The Together Open API is structured as a three-tier interface that allows varying degrees of access to the native infrastructure, as shown in Figure 116. The top tier represents the highest degree of constraint and the bottom tier the lowest degree of constraint.

Figure 116 Three-tier structure of the Together Open API



Following are descriptions for each component of the API:

- **IDE**

  The IDE generates custom outputs based on information contained in a Together model. It is a read-only interface, meaning that you can extract information from the model but can not change the model (accidentally or otherwise). The IDE group provides the functionality related to the model's representation in Together's IDE and interaction with the user. Each package of the IDE group has a description highlighting its area of applicability.

- **RWI**

  The RWI enables you to go deeper into the Together architecture. You can extract information from and write information to your models, and you can create extensions of Together's capabilities. RWI elements can represent more than packages, classes, and members. In a RWI model, elements can represent diagrams (such as class, use case, and sequence diagrams), links, notes, use cases, actors, states, and so on.

- **SCI**

  The SCI provides access to the source code level. This API is the most granular, enabling manipulation of single bytes. An SCI model is a set of sources organized into packages (for Java, `.class` files are allowed). The SCI packages represent the Java packages, which can be stored in `.zip` or `.jar` files.

Understanding modules

A *module* is a Java class that implements either IdeScript or the IdeStartup interface, or both. (For the location of these interfaces, see "Overview of the Together Open API" on page 461).

When implementing the IdeScript interface, you need to define the `run(IdeContext)` method; when implementing IdeStartup interface, you need to define the `autorun()` method.

Before you begin working with modules, it is helpful to understand the differences between modules, interfaces, and scripts within the context of Together:

- **Modules vs. Interfaces**

  The IdeStartup interface defines a module whose `autorun()` method is invoked automatically during Together's startup process. This method performs some module-specific actions such as registering a menu command item with an appropriate listener. After it finishes executing, the autorun() method will not be invoked again during the current Together user session.

  The IdeScript interface defines a module that can be invoked at any time, and any number of times during a user session by calling its `run(IdeContext)` method. An IdeContext instance (com.togethersoft.openapi.ide.IdeContext) being passed to the run method contains information about selection at the moment you ran the module.

- **Modules vs. scripts**

  The difference between a module and a script is mostly semantic. Both modules and scripts can use the Together API to interact with Together or access model information and process it. In Together documentation, *module* refers to a program written in Java, compiled by a Java compiler, and executed using the same Java virtual machine (JVM) as Together at runtime. *Script* refers to a scripting code that is interpreted by appropriate Together subsystems at runtime. Currently, Tcl, and JPython are supported as scripting languages, but this support may become deprecated in the future. For long-term compatibility, use Java.

Locating modules

A module defines its own subpackage with a name matching the name of the module and is located in the module package. For example:

```
package module.InsertTags;
import com.togethersoft.openapi.ide.IdeContext;
import com.togethersoft.openapi.ide.IdeScript;
public class InsertTags implements IdeScript{
   public void run(IdeContext context){
   }
}
```

Registering a module

To register a module, follow these steps:

- Compile a module. (For example, compile the InsertTags module described in the previous section.)
- Make a manifest file for your module. A manifest file is a simple text file which allows Together to identify the type of module.

The extension of a manifest file is `.def`, and the file name is the name of the module. It must be located in the classpath according to the package of the module. For example, the `InsertTags` manifest file is `$TGH$/modules/com/togethesoft/modules/inserttags/InsertTags.def`. For more information regarding `.def` files, see "Declaring a module in a .def file" on page 475

A manifest file consists of only one string, depending on the type of modules:

- For IdeStartup modules:

  ```
  Startup=true
  ```

- For IdeScript modules:

  ```
  Script=true
  ```

  (For example, since `InsertTags` is an IdeScript, it contains `Script=true`.)

Starting modules

To start a module, first determine the type of module you have:

- **Startup modules implementing the IdeStartup interface:** You can *not* manually run these modules. Together automatically invokes these modules during the session startup.
- **Modules implementing the IdeScript interface:** You can use the Modules tab of the Explorer pane. Navigate to the script, right-click on it, and choose **Run** from the right-click menu.

For related information, see "Running modules" on page 467. For information about other module types, see "Types of modules" on page 466.

Examples for *Hello World*

Following are two examples for writing *Hello World*:

- **Startup script:** This script writes the message at Together's startup process.

  ```
  package script.HelloWorldAutorun;

  import com.togethersoft.openapi.ide.IdeStartup;

  public class HelloWorldAutorun implements IdeStartup {
  ```

```
  public void autorun(){
    System.out.println("Hello world from HelloWorldAutorun script!");
  }
}
```

After compilation, create the file `$TGH$/modules/com/togethesoft/modules/HelloWorldAutorun/HelloWorldAutorun.def` with the line `Startup=true` in it.

- **Script implementing the *IdeScript* interface:**

```
package script.HelloWorld;

import com.togethersoft.openapi.ide.IdeContext;

import com.togethersoft.openapi.ide.IdeScript;

public class HelloWorld implements IdeScript {

  public void run(IdeContext context){

    System.out.println("Hello World!!!");

  }

}
```

After compilation, create the file `$TGH$/modules/com/togethesoft/modules/HelloWorld/HelloWorld.def` with the line `Script=true` in it. Now open a project and locate this script on the Modules tab of the Explorer. Right-click on it and choose the **Run** command.

Extending Together using modules

Together provides an open Java API that allows you to write modules, as discussed in the previous section. Modules can use modeling information from Together or extend Together's native capabilities. Many of Together's features are implemented as modules, and the architecture makes it possible for Together to include modules (also called *plug-ins*) developed by strategic partners or other third parties.

Module development can take one of the following two forms:

- **Compiled**: Compiled modules are written in Java and compiled using a Java compiler. Such modules use the same JVM as Together at runtime.
- **Written in Tcl (or JPython)**: Referred to as *scripts*, these modules are interpreted by appropriate Together subsystems at runtime.

Compiled modules deliver better performance and more depth, providing the full capabilities of Java.

Types of modules

A module is defined by its language, invocation time, and type of deployment. Generally, modules are categorized into the following groups:

- **User modules**: Added to the modules tree as icons and can be invoked by choosing Run from the module icon's right-click menu.
- **User with pre-initialization modules**: Similar to user modules, but the initialization method executes before the module; Assigned only to Java modules and assigned by default.
- **Startup modules**: Always run at Together startup and do not appear on the modules tree.
- **Activated (Deactivated) modules**: Combine the advantages of both User and Startup types; Added both to the modules tree and Activate/Deactivate Features dialog box invoked from the Tools menu.

To change the state of an activatable module, you can use the main menu instead of browsing the modules tree. The Activate/Deactivate Features dialog box shows a list of checkboxes for all available modules. Checking a box activates a module and unchecking deactivates the module.

The state of an activatable module is persistent between Together sessions. Once it is activated, the module behaves like a startup module.

Interfaces implemented by the modules

Modules are Java classes that implement the IdeScript, IdeStartup,or IdeActivatable interfaces. For more general information about modules, see "Understanding modules" on page 462.

Modules of the User type implement the IdeScript interface that provides the method `run()`. Startup modules implement the interface IdeStartup that provides the `autorun()` method. The modules that should provide the possibility of on-demand invocation and deactivation implement the interface IdeActivatable that extends IdeStartup with the shutdown method. If a module is supposed to be used both as a startup and user module, it should implement IdeScript and IdeStartup interfaces, while a startup module with the possibility of on-demand invocation should implement both IdeScript and IdeActivatable interfaces.

Viewing modules

Modules are stored in the subdirectories under `$TGH$\modules\com\togethersoft /\modules`. You can navigate to the modules using the Modules tab of the Explorer. This tab displays several folders:

- **System** contains all the modules that are part of Together itself.
- **Sample** contains sample modules and scripts (including some source files).

- **Early access** contains modules that are works in progress — whether not fully implemented, undocumented, or both.

Table 49 shows how Together represents modules in the Modules tab.

Table 49 Modules Tab

Icon	Description
	Java source file for a module. The source file is compiled by choosing Run from the right-click menu. Note that the compiler uses the same JDK that Together runs on, JDK 1.3.
	Compiled Java module.
	Tcl script. The Run command executes the script in interpreted mode.

Running modules

The modules from the System folder are incorporated into the Together menu system or run from dialogs. Some modules are activated through the list of Activatable Modules on the Tools menu. The others are available on the objects' right-click menus.

You can run any module (or script) from the Modules tab right-click menu. For modules or scripts that refer to or handle model information (as most do), you should open a Together project before running.

To run a module (or script):

1. Navigate to and select the module or script.
2. Right-click on the node and choose **Run**.

Guidelines for developing modules

Develop modules that access model information in order to:

- Generate model and code documentation in custom formats
- Export to different file formats
- Develop patterns and experts

If you create your own modules and save them to the relevant folders under `$TGH$/modules/com/togethersoft/modules`, they display on the Modules tab and can run from there. Add commands to the menu system for launching your own modules by creating a Tool definition in the Options dialog (Tools | Options | Default Level | Tools), or by customizing the files `$TGH$/config/menu.config` and or `$TGH$/config/action.config`.

Applying naming conventions

Together modules are classes implementing either the IdeScript or IdeStartup interface, or both. The names of modules use mixed case (for example, GenerateDocumentation, ImportFiles, and so on). Table 50 provides guidelines for

Table 50 Naming conventions for modules

Item	Convention	Example	Benefit
Method	Methods should be named using a full English description, using mixed case with the first letter of any non-initial word capitalized. It is also common practice for the first word of a method name to be an active verb.	• processModel() • printCurrentDiagram() • recheckAllNames() • iterateCurrentNode()	Results in methods whose purpose can be determined by name, making it easier to understand and maintain the code.
Getter	Getters are methods that return the value of an attribute. Use “get” as a prefix to the name of the attribute; for a Boolean attribute, use the prefix “is”	• getFirstName() • getAccountNumber() • getLostEh() • isPersistent() • isAtEnd()	Indicates that a method returns an attribute of an object; or, for boolean getters, that it returns true or false.
Setter	Setters are methods that modify the values of an attribute. Use “set” as a prefix to the name of the attribute, regardless of the attribute type.	• setFirstName(String aName) • setAccountNumber(int anAccountNumber) • setReasonableGoals(Vector newGoals) • setPersistent(boolean isPersistent) • setAtEnd(boolean isAtEnd)	Indicates that a method sets the value of an attribute of an object.
Attribute	Use a complete English descriptor to name your attributes. Attributes that are collections such as arrays or vectors should be given names that are plural to indicate that they represent multiple values.		Indicates what the attribute represents.

naming conventions.

Documenting the module

In the interest of brevity, use phrases instead of complete sentences when documenting a module. In particular, use brief descriptions for the initial summary and `@param` tag descriptions. This section provides additional guidelines for documenting modules.

Use 3rd person (descriptive) rather than 2nd person (prescriptive) as follows:

```
Gets the label. (preferred)
Get the label. (avoid)
```

Method descriptions begin with a verb phrase. A method implements an operation, so it usually starts with a verb phrase:

```
Gets the label of this button.  (preferred)
This method gets the label of this button.  (avoid)
```

Add a description beyond the name. The best names are “self-documenting,” meaning that they indicate what the method does. If a comment repeats the method's name in sentence form, it is not providing more information. The ideal comment goes beyond those words and should always reward you with some bit of information that was not immediately obvious from the name.

Avoid - The description below says nothing beyond what you know from reading the method name. The words “set”, “tool”, “tip”, and “text” are simply repeated in a sentence.

```
/**
 * Sets the tool tip text.
 *
 * @param text The text of the tool tip.
 */
public void setToolTipText(String text) {
```

Preferred - The following description more completely defines what a tool tip is, in the larger context of registering and being displayed in response to the cursor.

```
/**
 * Registers the text to display in a tool tip. The text
 * displays when the cursor lingers over the component.
 *
 * @param text The string to display. If the text is null,
 * the tool tip is turned off for this component.
 */
public void setToolTipText(String text) {
```

Order of Tags

Include tags in the following order:

```
* @author  (classes and interfaces only, required)
* @version  (classes and interfaces only, required)
* @param  (methods and constructors only)
* @return  (methods only)
* @exception  (@throws is a synonym added in Javadoc 1.2)
* @see
* @since
* @deprecated
```

Deploying the module

A module defines its own subpackage with the name matching the name of the module, located in the `modules` package. For example:

```
package com.togethersoft.modules.InsertTags;
import com.togethersoft.openapi.ide.IdeContext;
import com.togethersoft.openapi.ide.IdeScript;
public class InsertTags implements IdeScript{
   public void run(IdeContext context){
   }
}
```

Tutorial: Writing and deploying modules

This section describes the full process of writing and deploying extension modules. It includes instructions for writing a simple module and explains how to set-up Together to recognize it.

There are two major steps in module development. First, write the module source code. Second, declare the module. Confirm that module declaration and source code are consistent. For example, if a module is declared OnDemand, but the main class of this module implements the IdeScript interface, then the module will not work.

Source code for the module

The module described in this exercise will run both as a startup and as a regular Together module. Thus, you need to implement both of the following interfaces: `com.togethersoft.openapi.ide.IdeScript and com.togethersoft.openapi.ide.IdeStart`

To output messages to the Together Message pane, use the interfaces:

`com.togethersoft.openapi.ide.message.IdeMessageManagerAccess`

and

`com.togethersoft.openapi.ide.message.IdeMessageType`

(Refer to the API documentation in `$TGH$/doc/api`.)

Important Each new module must define its own package in the `com.togethersoft.modules.InsertTags` package.

To follow this exercise, create the following directory:

TGH\modules\com\togethersoft\modules\myFirstModule

Create the source file listed below, and name the file: `MyFirstModule.java`

```
package com.togethersoft.modules.myFirstModule;

import com.togethersoft.openapi.ide.IdeContext;
import com.togethersoft.openapi.ide.IdeScript;
import com.togethersoft.openapi.ide.IdeStartup;
import com.togethersoft.openapi.ide.message.IdeMessageManagerAccess;
import com.togethersoft.openapi.ide.message.IdeMessageType;

public class MyFirstModule implements IdeScript, IdeStartup {
  public void run(IdeContext context) {
    //Open the message pane
    IdeMessageManagerAccess.getMessageManager().setPaneVisible(true);
    //Regular modules perform this method.
    IdeMessageManagerAccess.printMessage(IdeMessageType.INFORMATION,
      "This is my first module called as a regular Together module.");
  }

  //Startup modules perform this method.
  public void autorun() {
    //This will appear in the Message pane when Together is loaded.
    IdeMessageManagerAccess.printMessage(IdeMessageType.INFORMATION,
      "This is my first module.  It was called as a startup module.");
    //This will appear in the Console window at the startup of Together.
    System.out.println("This is my first module, called a startup
      module.");
  }
```

```
}
```

Declaring a module

The Modules tab of the Explorer does not show a module until you declare the module. The most important issue to consider when writing a module is the set of properties that will be used for module declaration. Module declaration consists of using a text manifest file to declare module type (by invocation time), name, location in the module tree, dependencies, additional libraries, and other important properties.

A manifest file is either a `Manifest.mf` file or a `.def` file. Generally, the `Manifest.mf` file applies to modules written in Java. Use the `.def` file if you used Tcl to write your module.

Declaring a module in a Manifest.mf file

For modules written in Java, use the `Manifest.mf` file to declare your module. If you used Tcl, see "Declaring a module in a .def file" on page 475.

Following is a list of properties available for the modules.

- **Main-Class (MainClassName):** This property is mandatory for all types of JAVA modules declared in the Manifest.mf file (except for JAVA modules declared in the `*.def` files). The name in brackets is the alternative property name that should be used when the module is declared in a `*.def` file.

  The class specified as a value of this property should exist and should implement the Java interface that corresponds to its type (`IdeScript`, `IdeStartup`, or `IdeActivatable`).

  Example of property declaration in `Manifest.mf` file:

  ```
  Main-Class: com.togethersoft.modules.ejb.EJB
  ```

  An Example of the analogous property declaration in `*.def` file:

  ```
  MainClassName=com.togethersoft.modules.helloworld.HelloJava
  ```

- **Name:** This property specifies the module name to be used in the module tree, in the **Tools | Activate/Deactivate Features** dialog box.

  Example:

  ```
  Main-Class: com.togethersoft.modules.ejb.EJB

  Name: EJB Support
  ```

- **Folder:** The `Folder` property specifies the module location in the module tree. It makes sense only for User, User with pre-initialization, and Activatable module types. If this option is not specified, the Early Access folder is used by default. The value of this property is case-sensitive.

  Example:

```
Name: Test Export

Folder: "System/Test/Test Export"
```

As a result of the declaration above, the module Test will be found in the modules tree under All modules - System - Test, and the module node(s) will be named, Test Export.

- **Time:** This property defines the module type by invocation time. It is mandatory for the Java modules. The value is case-sensitive.

  Examples:

  ```
  Time: User
  ```

  Declares the module as User or User with pre-initialization (depends on the main class of the module).

  ```
  Time: OnDemand
  ```

  Declares the module as Activatable.

  ```
  Time: Startup
  ```

  Declares the module as Startup.

- **Script, Startup, OnDemand:** These properties are an alternative for the Time property and are now deprecated.
- **ActivatedByDefault:** This option can be used for Activatable modules only. Available values are true and false. By default, this option is false. Setting this option to true means that if an Activatable module was not explicitly deactivated, it is in the activated state.
- **Services:** This property should be used when the module can be used by some other modules. Service names (case-sensitive) should be specified as property values. Other modules will in turn specify these names as the names of the services they depend on.

  Example:

  ```
  Name: EJB

  Time: Startup

  Main-Class: com.togethersoft.modules.ejb.EJB

  Services: EJB, EJBClassDiagram
  ```

  This declares that services `EJB` and `EJBClassDiagram` are available and modules that depend on these services will be loaded.

- **DependsOn:** This property is closely related to the previous property. It declares services that should be available to load this module. If at least one of the specified services is not available, the module does not load, and Together generates an error message.

  For example (module "A" depends on module "B"):

```
Name: |A|
Main-Class: com.togethersoft.modules.a.A
Time: User
DependsOn: B
Name: |B|
Main-Class: com.togethersoft.modules.b.B
Time: User
Services: B, ExtendedB
```

If module B is not deployed, module A does not load.

- **ClassPath:** This property specifies additional class libraries (jar archives or folders) that are used by this module. The value of this property can be either the fully qualified path or the relative path to this module home directory. It is also possible to use "`$TGH$`" macros.

  In the following example, classes from the libraries are available for the module at runtime:

  ```
  Class-Path: libInHomeFolder.jar;$TGH$\lib\lib.jar;C:\libs\lib1.jar
  ```

- **HelpFile:** This property adds the Help command to the module's right-click menu in the module tree. The path to the help file must be fully qualified.

  Example:

  ```
  Main-Class: com.togethersoft.modules.test.export
  Name: Test
  HelpFile: $TGH$\module\com\togethersoft\modules\test\doc\index.html
  ```

- **Options:** This option adds the Options command to the right-click menu of the module. The value of this property is the name of the properties page, the same as used in the `IdeConfigManager.showConfigEditor` method (for more information, refer to `$TGH$\doc\api`).

  For example:

  ```
  Main-Class: com.togethersoft.modules.test.export
  Time: User
  Name: Test Export
  Options: General
  ```

  In the example above, the General properties page will open when using the Options command from the module's right-click menu. For Activatable modules, this command displays only when the module is activated.

Note Only the default level options can be shown in this way. If an options page with the specified name does not exist, it displays an empty dialog.

- **Hidden:** Enables you to hide (true) or show (false) User or Activatable modules in the module tree.

Declaring a module in a .def file

This section includes instructions for declaring a module in a `*.def` file. Generally, use the `*.def` file to declare a module written in Tcl. Otherwise, declare your module in the `Manifest.mf` file.

Specifying whether a module is a startup module

Startup modules are modules that load when Together starts. If your module is intended to be a startup module, the manifest must contain the line:

```
Startup = true
```

For regular (i.e., non-startup) modules the manifest must contain the line:

```
Script = true
```

This should be the first line of the manifest file. Each manifest file must have a minimum of one line (either `Startup = true` or `Script = true`). You can add both lines if you want the module to run automatically on startup, and if you want to be able to run the module during the Together session.

The manifest file can contain additional lines that provide information for Together so that the module appears in and is accessible from the Together user interface.

Defining a visible name for the module

You can specify a visible name for the module by adding this line to the manifest file:

```
Name = "Visible Name"
```

The string after the equal sign is displayed as the name of the module on the Modules tab of the Explorer. In this example exercise, specify a Visible Name.

Specifying the location of the Modules tab

You can specify where your module should appear in the Modules tab of the Explorer. For example, you can display it in the existing `Sample` folder, or you can create a new folder. To specify the Modules tab location, add this line to the manifest file:

```
Folder = "<Folder name>"
```

`"<Folder name>"` can be anything you want.

Examples:

`Folder = "Sample"` (module appears in the existing Sample folder node of the Modules tab)

`Folder = "My first module"` (module appears in a new folder My first module on the Modules tab)

If a folder is not defined in the `*.def` file, then the module is placed under the `Early Access` folder by default.

Adding the module name to menus for classes, interfaces, and members

You can display the module in the Modules submenu of the right-click menu for classes, interfaces, attributes, and operations. Add the line:

```
PopupMenuItem = true
```

Adding the module name to menus of elements with specific shapetype

Display the module in the Modules submenu on the right-click menus of elements having a specific shapetype For an explanation of this term, refer to the documentation for the property `RwiProperty.SHAPE_TYPE` in the API documentation.

Briefly, by optional addition of a constraint using the shapetype parameter, you can display the module in the Modules submenu only for operations, only for attributes, or only for classes and interfaces. Two lines are required to accomplish this:

```
PopupMenuItem = true
PopupConstraints="shapeType=<shapetype identifier>"
```

Examples:

- Only for operations:

  ```
  PopupMenuItem = true
  PopupConstraints = "shapeType=Operation"
  ```

- Only for attributes:

  ```
  PopupMenuItem = true
  PopupConstraints = "shapeType=Attributes"
  ```

- Only for classes and interfaces:

  ```
  PopupMenuItem = true
  PopupConstraints = "shapeType=Class"
  ```

Note Since classes and interfaces have the same shapetype (i.e., Class), you cannot selectively limit the display of a module to interfaces using the manifest file. Instead, refer to the module in `$TGH$\modules\com\togethersoft / \modules\tutorial` for information.

- Only for actors:

  ```
  PopupMenuItem = true
  PopupConstraints = "shapeType=Actor"
  ```

Tip The possible shapetypes are defined in the `RwiShapeType` interface. For more information, refer to the API documentation located in the `$TGH$\doc\api` directory.

Rules for the manifest file

The manifest file must satisfy the following conditions:

- The manifest file name must be exactly the name of the module's main class (the class implementing `IdeStartup` or `IdeScript` interfaces).
- The manifest file location must be somewhere under the directories specified in the file `$TGH$/config/scriptloader.config`.

This file defines two possible root directories for manifest files:

1. `$TGH$\modules\com\togethersoft\modules`
2. `$TGH$\classes\com\togethersoft\modules`

Since MyFirstModule can be used as both kinds of modules (startup and runnable during session), we will use both of the module declaration lines (`Script = true`, `Startup = true`). Open your text editor and create the manifest file now. Add the following lines:

```
Script = true

Startup = true

Folder = "Sample/MyFirstModuleFolder"

Name = "My First Module"
```

Save the file as:

`$TGH$\modules\com\togethersoft\modules\myFirstModule\MyFirstModule.def`

Compiling the module

Use JBuilder or your favorite Java compiler to compile the module source code. Make sure to include the `$TGH$\lib\openapi.jar` file containing API classes in your compiler's classpath.

Storing the compiled class

Keep the module's `*.class` file(s) in the same directory where the manifest file and sources reside. If you keep `*.class` files somewhere else, make sure that their relative paths match the path of the manifest file counting from one of the following two directories:

`$TGH$\modules\com\togethersoft\modules`

`$TGH$\classes\com\togethersoft\modules`

For example, if your manifest file is:
`$TGH$\modules\com\togethersoft\modules\coolModule\CoolModule.def`,

then your `*.class` files must be somewhere in the classpath under the directory:
`com\togethersoft\modules\coolModule\`.

Especially for your first modules, keep `*.class` files in the same directory as the `manifest.mf` (or `*.def`) file and the sources. For example, you can keep them together in a separate directory under `$TGH$\modules\com\togethersoft\modules`.

Evaluating the results

Once you have compiled the module, run Together. If the console window is open, you should see the module's startup message to indicate when it was called at the startup and when it performed the autorun method.

Note Although the autorun method contains two output commands, only one console message appears. When Together is loading, its message pane is not visible, so all the messages written to it at the startup are displayed only after Together finishes loading.

When Together is loaded, select the Modules tab in the Explorer. Expand the `All modules` folder. You should see the `MyFirstModuleFolder` node with the `My First Module` files. One file is the module's source, and the other is the compiled `.class` file. Select either file, right-click, and choose **Run**. When right clicking the source file, the Run command will cause Together to compile and run the module. To use your home-brewed compilation results, select the.class file. If the message pane is open, you should see the module's message.

It is strongly recommended to sequentially comment out each line of the manifest file to see how this makes the module work in only one mode. Try it:

```
Script = true
;Startup = true
Folder = "Sample/MyFirstModuleFolder"
Name = "My First Module"
```

and

```
;Script = true
Startup = true
Folder = "Sample/MyFirstModuleFolder"
Name = "My First Module"
```

Note Restart Together after each change in the manifest file to see your changes.

Troubleshooting your module

The declaration and `.java` files of the module described in this document are not provided with Together. You need to create your own first module. The process of creating, deploying, and running your own module will allow you to develop future modules more efficiently.

After you successfully deploy and run this example, look through the sample modules located in the `$TGH$\modules\com\togethersoft\modules` directory. Pay attention to the `Tutorial` folder, which contains a set of sample modules demonstrating the usage of the Together open API.

Try to take something from the first lesson modules in the tutorial folder and use it in a new module (or simply rename a tutorial lesson to be your second module).

If you receive compilation errors, check that you have:

- Imported all of the required API interfaces/classes.
- Added all the required libraries to the compiler's classpath.
- Used API methods properly. See API documentation in `$TGH$/doc/api`.

Other common problems include:

- Your code compiles without errors, but you can not locate your module on the Modules tab.

  Check that you created a manifest file and placed it under the directories specified by the `scriptloader.config` file.

  By default, these are specified as:

  ```
  $TGH$\modules\com\togethersoft\modules
  ```

  ```
  $TGH$\classes\com\togethersoft\modules
  ```

- Your code compiles without errors, and the `.def` file is located correctly, but you still can not locate your module on the Modules tab
  - Check for spelling errors in the lines `Startup = true` or `Script = true` in the `.def` file.
  - Check that the name of the manifest file is not identical to the name of the module. In such a case, Together's Message pane will display an error such as:

    "The manifest file `c:\together\modules\com\togethersoft\modules / \coolModule\CoolScript.def` exists but Together either cannot find or cannot read its associated module files."

- Your code compiles without errors and you see the module's folder on the Modules tab, but there are only Java sources (assuming you want to see and run the compiled module)

  Check that the module's `.class` files are located in the right place so that Together can find them. Keep the `.class` files in the same location as the `.def` file.

- Your code compiles without errors, and you see the module's compiled class on the Modules tab, but your module does not appear to function properly.

Mark the start and finish of each of your modules (regular and startup) with appropriate messages in the Message pane. This clarifies whether your module was actually running or not. For example:

```
IdeMessageManagerAccess.printMessage
(IdeMessageType.INFORMATION,"MyCoolModule:started");//start
IdeMessageManagerAccess.printMessage
(IdeMessageType.INFORMATION,"MyCoolModule:finished");//finish
```

Accessing more API documentation

The information in this chapter is limited to an introduction of the Together Open API. For a more detailed technical reference, refer to the API documentation listed in:
`$TGH$/doc/api/index.html`. To learn how to use the Open API documentation, refer `to $TGH$/doc/api/help-doc.html`.

Appendix

A

Configuring Together for Multiple Users

The information in this chapter is specifically for the site administrator responsible for setting up Together for multiple users.

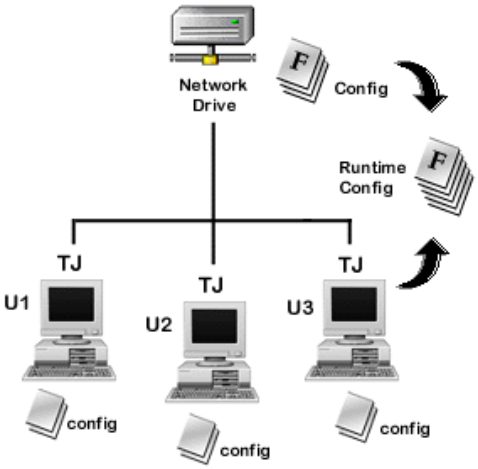
This chapter includes the following topics:

- "Overview of a shared multi-user configuration" on page 481.
- "Setting-up multiple workstation installations" on page 482.
- "Adding new configuration levels to predefined levels" on page 484.

Overview of a shared multi-user configuration

Together uses a tiered, hierarchical approach to configuring properties, meaning that as the Together administrator, you can configure options at three pre-defined levels: Default, Project, and Diagram. This enables you to provide a set of configuration properties (such as code-generation rules) for all users of the installation. You can specify which options users can not override, and which options users can configure with their own settings.

Figure 117 Together running on individual workstations



Setting-up multiple workstation installations

Multiple users with their own copies of Together running on separate workstations can access a shared multi-level configuration. As the Together administrator, you can store configuration properties for the group in a centrally accessible location on the network. Users launch Together from a batch or command file, specifying the path to the central configuration using the `-config.path` command-line switch.

Note In addition to the predefined configuration levels of default, project, and diagram, you can add new configuration levels to Together that users can access. The instructions that follow include a reference to this option.

To set up a multi-workstation shared configuration for a shared installation, complete these steps:

1. Install Together on your (the Together administrator's) workstation. Make a back-up copy of the original `./config` directory of the Together installation.
2. Run Together on your workstation.
3. If additional configuration levels are needed, create these as described in "Adding new configuration levels to predefined levels" on page 484.
4. Edit configuration options as required by choosing **Tools | Options |** ***applicable_level***, and clicking the **Levels >>** button. Mark individual options as "final" at the desired level for those settings that you do not want individual users to override from a lower level.

   See Chapter 3, "Configuring Options" for detailed instructions for setting-up options.

5. Copy the modified `./config` directory (plus any additional directories created for added config levels) to a shared network location (for example, `<server>/tg_shared_config`) that is accessible to all Together users who use this configuration.

Creating the start-up pointer file

To centrally share the entire configuration, you need to create a pointer file to define the centralized location and override Together's hard-coded configuration locations. Once you create the pointer file and define its location(s), you need to reference the pointer file in the `-config.path` switch when you start Together from the command line (or from a batch or command file). The most convenient way to create the pointer file is to let Together generate a copy of its own defaults.

To generate a default pointer file:

1. Run the Together launcher for your OS and pass:

   `-debug.config.saveDefaultPath=<path to file>`
   in the command line. For example:

   ```
   Together.exe -debug.config.saveDefaultPath=c:/TogetherSoft/Together6.0/lib/pointer.config
   ```

2. Together starts up and writes the specified file to disk. You can then close Together.

The file name used in the example is just a suggestion. You can use any name that does not conflict with Together property file names. You can store the pointer file anywhere, but it is probably most convenient to keep it in the `./ lib` directory of each local installation. Immediately after you generate it, the `pointer.config` file will contain the following lines:

```
config.level.$internal.group = basic
config.level.$internal.0 = $TOGETHER_LIB$/internal.config
config.level.$internal.1 = $TOGETHER_LIB$/path.config
config.level.$default.group = session
config.level.$default.0 = $TOGETHER_CONFIG$/*.config
config.level.$default.write = $TOGETHER_CONFIG$/changes.config
config.level.$commandLine.group = basic
config.level.$project.group = project
config.level.$workspace.group = project
config.level.$workspace.0 = $PROJECT_DIR$/*.config
config.level.$workspace.write = $PROJECT_DIR$/$PROJECT_NAME$.tws
```

You need to edit the highlighted (in **bold**) lines to point to your shared configuration. Note that paths must reference mapped drives (for example, this path will not work: `\\appserver\tg_shared_config\*.config`). So if your shared configuration is on `appserver` which is locally mapped to drive `s:`, then your edited lines should look like this (Windows style paths shown):

```
config.level.$default.0 = s:\tg_shared_config\*.config
config.level.$default.write = s:\tg_shared_config\changes.config
```

Launching via the command line and pointer file

You should create for each user, or instruct users to create on their workstations, a start-up batch or command file that launches Together locally from the command line using the `-config.path` switch to specify the path to the pointer file. In Windows, the command would look something like this:

```
c:\Together\bin\Together.exe -config.path=c:\Together\lib\pointer.config
```

Note If you are using the Together floating license server, Together may not use the `pointer.config` file to find the `server.config` file. As a workaround, you need to start Together using this command:

```
together.exe -license.config.path=<config_dir> -config.path=$TGH$\lib\
pointer.config
```

Adding new configuration levels to predefined levels

You can add up to three additional configuration levels to the pre-defined levels of Default, Project, and Diagram. New levels must be inserted above the installation-wide Default level in the Options dialog order.

For example, a Corporate level could be added to enforce certain configuration settings across the enterprise. You could define the File prologue option in Source Code options so that all generated source code files contain the corporate copyright. Marking this final at the Corporate level prevents changes from lower levels.

To add configuration levels, follow steps:

1. Copy the contents of the `$TOGETHER_HOME$/config` directory to another location to create a separate set of configuration properties for the new level. Possible locations include a shared network location (as described in the previous section) or a local directory in your Together installation (e.g. `config/corporate`).
2. Create a `path.config` file in the `./lib` directory and point it to the directory you set up for the new levels. Together searches this file and loads additional configuration levels from the locations specified. If the location in the previous step is shared, be sure to perform this second step on all the local machines that need to share the configuration.

Adding new configuration directories to a shared location

If you are creating the new levels for a configuration that will be shared from a central location, you have to create a set of configuration properties files for each new configuration level.

To create configuration properties directories for a shared location:

1. Copy the `./config` directory of your Together installation to the shared location and rename it to `tg_shared_config` (or some other meaningful name).
2. For each configuration level you plan to add, copy the `./config` directory of your Together installation to a new subdirectory of `tg_shared_config`. Rename each new subdirectory with the name of the level it represents, such as `tg_shared_config/corporate`.

Tip Create the subdirectories under your local config directory, test that you have defined the new levels correctly, and then copy the config structure to the shared location and modify the levels from there.

Adding new properties directories to a local installation

If you are creating the new levels for a configuration on a local installation, you have to copy the `./config` directory of your Together installation to another location and rename it with the name of the level it represents such as `./config/corporate`.

For example, to create a Corporate configuration level, first copy everything in the config directory to some other directory, such as `$TOGETHER_HOME$/config/corporate`.

Creating the path.config file

Before you create the path.config file, make sure that it does *not* already exist. If you are setting up a shared configuration, you need to place a copy of this file in the Together installation of *each* user.

Use a text editor to create the file `$TOGETHER_HOME$/lib/path.config`.

Add the following lines to the file:

```
=== path.config ===
 optionsEditor.level.$corporate.name = Corporate
   optionsEditor.level.$corporate.visible = true
   config.level.$corporate.name = Corporate
   config.level.$corporate.visible = true
   config.level.$corporate.0 = <path to this level>
   === path.config ===
```

Replace `<path to this level>` with the path to the properties files for the level you are defining, such as `$TOGETHER_CONFIG/corporate/*.config`.

Note `config.level.<level name>.<number> = <path>` defines the source for config files. For shared locations, use the full path. For local paths, specify the full path if outside your Together installation; within that structure you can substitute an appropriate Together System Macro (such as `$TOGETHER_LIB$, $TOGETHER_CONFIG$, $TOGETHER_HOME$)` as part of the path specification.

If you are adding more than one level, you should add some additional lines for each one. For example, to add a Division level:

```
...
optionsEditor.level.$division.name = Division
optionsEditor.level.$division.visible = true
config.level.$division.name = Division
config.level.$division.visible = true
config.level.$division.1 = $TOGETHER_CONFIG$/division /*.config
```

Viewing added configuration levels

To verify that you have correctly added a new level, run Together, open the Options dialog from the main menu, and click the Levels button. (If you created the new levels in a shared location, you will need to launch Together from the command line specifying the path to the configuration.)

In the case of our Corporate example, you should see the new Corporate level in the drop-down list of available levels.

In this way you can configure local installations to include company-wide configurations. Company-wide levels will be configured through `path.config`, and the installation-wide Default level will become user-specific.

Modifying default configuration levels

To get a server installation with user-specific configuration, the entire configuration order should be redefined. Together has the command line parameter

`-config.path=<path to file which defines all levels>.`
This option overrides hard-coded levels configuration. To get the hard-coded settings written to a file, pass

`-debug.config.saveDefaultPath=<path to file>`
in the command line. You will get a file, which can be used in the `-config.path` option to set the default behavior. This file can be modified to add new levels.

To make a new level visible in the Options dialog, you should include additional lines in the config file (such as misc.config):

```
optionsEditor.level.$corporate.name = Corporate
optionsEditor.level.$corporate.visible = true
optionsEditor.level.$department.name = Department
optionsEditor.level.$department.visible = true
```

`[...].name` is optional. If absent, the internal name will be shown (“corporate”).

`[...].visible` is necessary, because the level is not visible by default.

Appendix

B

Commands, Macros, and Keyboard Shortcuts

This appendix includes the following topics:

- "Command Line Parameters" on page 489.
- "Parameters for the Together.exe Launcher" on page 493.
- "System Macros" on page 494.
- "Template Macros" on page 498.
- "Keyboard shortcuts" on page 499.

Command Line Parameters

This section provides the command line parameters for Together and includes the following topics:

- "Basic command line syntax" on page 489.
- "Using the Windows launcher" on page 491.
- "Invoking the Together main class" on page 491.

Basic command line syntax

Table 51 lists the command line syntax for:

```
Together_starter [Options] [ProjectFile].
```

Table 51 Command line syntax

`Together_starter`	Command that starts Together. On the Windows platform, Together has the following launchers: • `Together.exe` with disabled console output, • `TogetherCon.exe`, that uses the existing console, or opens a new console for output. The console version is preferable for automated doc generation. In some cases, when graphic output is advisable in the console version, you can use the option `-gui`. For other platforms, you can create `Together.sh`, `Together.cmd`, etc. depending on the operating system. Alternatively, this can be a complicated command that calls your Java VM, specifying parameters for it, followed by the Together main class name `(com.togethersoft.together.Main)` followed by any parameters for that class. See "Invoking the Together main class" on page 491 for details.

`Options`	One or more concordant options, starting with hyphen. If an option requires a value, you can use either "=" (equal) or ":" (colon) symbols after the option's name. For example you can type:
	
	`-script=com.togethersoft.modules.helloworld.HelloJava`
	
	or
	
	`-script:com.togethersoft.modules.helloworld.HelloJava`
	
	Note: Use the colon (:) symbol under Windows, since "=" is not recognized by the Windows command line interpreter.

Table 51 Command line syntax

`ProjectFile`	Path to a Together or JBuilder project file to be opened. The file name must have either a `.tpr` or `.jpx` extension. Optional for invoking modules that do not need to access model information from a specific project. **Tip** In UNIX systems, the launcher takes spaces for parameters separators. Thus, the project file names, which contain spaces, are treated as the different file names. To avoid this problem, the spaces should be preceded with the backslash characters. For example, if the project file name is Java Project.tpr, it should be written as Java\ Project.tpr.
`ProjectFile`	Path to a Together project file to be opened. The file name must have `.tpr` extension. Optional for invoking modules that do not need to access model information from a specific project. **Tip** In UNIX systems, the launcher takes spaces for parameters separators. Thus, the project file names, which contain spaces, are treated as the different file names. To avoid this problem, the spaces should be preceded with the backslash characters. For example, if the project file name is Java Project.tpr, it should be written as Java\ Project.tpr.

Using the Windows launcher

When Together is started using one of the Window-specific launchers (`Together.exe`, `TogetherCon.exe`, or `umldoc.exe`), the Environment variables and JVM parameters are taken from the `Together.bat` file (`$TGH$\bin\Together.bat`). If there is no built-in Java-machine in the Together installation, the launchers make use of the JVM that corresponds to the value of JDK variable as defined in Together.bat file.

For complete parameters of `Together.exe` see "Parameters for the Together.exe Launcher" on page 493.

Invoking the Together main class

Under all supported operating systems, you can invoke the Together main class directly, specifying any desired or necessary parameters. The name of this class is:

```
com.togethersoft.together.Main
```

Since this can require a complex command that calls your Java VM, specifying parameters for it, followed by the Together main class name, followed by any parameters for it, you will probably prefer to create batch/command/shell script files for this kind of startup. You can find examples of such files in several platform-specific subdirectories in `$TGH$/bin`. Use these files directly, configuring as necessary for your system, or use them as models to create your own launcher files.

Table 52 lists the parameters for main class.

Table 52 Parameters of the main class.

`<Project.tpr>`	Specifies the fully qualified name of Together project file that will automatically open on start-up.
`<Project.tpr>` `<Project.jpx>`	Specifies the fully qualified name of Together or JBuilder project file that will automatically open on start-up.
`-config.path:<path>`	Tells Together to search <path> for configuration properties instead of the default hard-coded location. For information on usage see Creating a shared multi-user configuration
`-script: <module name>`	Automatically runs a module after opening the specified project. If module name is specified without the full path, Together searches for the module in the default locations (depending on the file extension). For more information, see Working with modules. **Note** when a script is launched, the simplified initialization of Together is performed. It means that all output is forwarded to the console, and GUI dialogs are disabled. If you wish to launch a script with full initialization of Together and GUI support, make sure to use `-gui` option in the command line.
`-version`	Types Together version and build number, and exits. *
`-help`	Types parameters of Together's main class, and exits. *
* This option is only used if console is enabled.	

Command-line examples

This section provides command line examples for documentation generation and running modules at start-up.

Documentation generation

If you have a nightly or other periodic automated build process, you can update your documentation as part of it by having your process script call Together's HTML or RTF documentation generation facility through the command line

interface. Together provides the possibility of generating documentation without actually opening Together window. Refer to the section Automated Doc Generation for details and examples.

Running modules at start-up

The following command line starts Together and executes the `Hello_World` module. (Note that the `scriptloader.config` file must be configured in accordance to your Java environment). The project file name parameter is optional in this case, since the `Hello_World` module does not access project information. Note that Java files and Java classes should be specified with the proper case of letters (Java is a case-sensitive language).

```
cd %TOGETHER_HOME%\bin <Enter>
```

```
TogetherCon -script:com.togethersoft.modules.helloworld.HelloJava samples\
java\CashSales\CashSales.tpr
```

Parameters for the Together.exe Launcher

The `Together.exe` launcher is provided for use under Win32 operating systems. Parameters of this executable file may be used when invoking it from the command prompt.

When invoking `Together.exe` from the Windows command line (or batch files), specify its parameters before any other parameters. You can see the complete list of Together.exe parameters by running it from the Windows Command prompt using the
`-h` option.

Parameters for virtual machine preferences

Together includes the parameters `-builtin` and `-sun13`. These enable you to specify the preferred virtual machine (VM) in cases where more than one is installed. By default, the Windows launcher first looks for the Microsoft JVM and uses it to run Together. Specifying `-sun` tells the launcher to prefer the Sun JVM (assuming it is present). If the preferred VM is not found on your computer, then another one is used. If no VM is found, the launcher returns an error message. Table 53 lists valid options for the following:

```
Together.exe [options] [-c[class_name]] [parameters]
```

Table 53 Options for Together.exe

Option	Description
`-?,-h,-help`	Print this usage message and exit
`-con,-nocon`	Show/don't show console (default without new console)
`-cmd`	Print launched command line
`-nowarn`	Do not show warning messages

Table 53 Options for Together.exe (continued)

-full	Load all *.zip/jar from TOGETHER\lib
-profile	Display time for common operation
-verbosegc	Print when garbage collection occurs
-noclassgc	Disable class garbage collection
-ms<number>	Initial java heap size (default 64m)
-mx<number>	Maximum java heap size (default 512m)
-D<name>=<value>	Set system property (or -d:<name>=<value>)
-Xbootclasspath <path>	Set bootclasspath to <path> (JDK 1.3)
-classpath <path>	Set classpath to <path>
-cp <path>	Prepend <path> to classpath (same as -cp:p <path>)
-cp:a <path>	Append <path> to classpath
-verify,-noverify	Verify/don't verify classes when loaded
-builtin	Prefer built-in Java VM (Sun JDK) to other Java VM
-sun13	Prefer Sun JDK 1.3 VM to other Java VM. **Important:** Together supports only JDK 1.3.
-nosystemcheck	Do not check system memory size
-J<runtime flag>	Pass argument to the Java interpreter
-c<class_name>	Class name to run (default com.togethersoft.together.Main)

The rest of command line after -c or unknown parameter prefix is class_name or its parameters. By default, JDK 1.3 (installed with Together on Windows platforms) is used. For example:

```
Together -sun13 -ms16m param1 param2 param3
```

System Macros

System macros are shorthand notations for lengthy path specifications that you can use for configuring properties, scripting or other tasks. Together knows how to expand the shorthand and make the proper reference.

Use macro references when completing customization tasks in the Tools category of the Options dialog. Table 54 uses **bold** text to indicate references to the Tools category.

Table 54 System macros

Macro	Description
TGH	Contains the full path to your Together's installation. For example (c:\Together).
$TOGETHER_HOME$	Identical to TGH

Table 54 System macros (continued)

$SYSTEMJVM$	Contains the call of the installed Java VM, along with the value of the system's CLASSPATH environment variable. For example: `c:\jdk1.2\bin\java.exe -cp %CLASSPATH%` (for Sun VM) `c:\WINNT\jview.exe /cp %CLASSPATH%` (for Microsoft VM) You can use this macro in the "Java VM" option and add additional directories/archive files after a semicolon (in Windows) or a colon (in UNIX). For example: $SYSTEMJVM$;c:\MyClassesDir (Windows)
$CLASSPATH$	Contains the value of the system's CLASSPATH environment variable.
$CLASSPATH_JVM$	Contains the path to the rt.jar file.
$CLASSPATH_PROJECT$	Contains the paths to all the packages in the project.
$SOURCEPATH$	Contains the paths to all the writable packages in the project.
$DESTINATION$	Contains the value of the *Destination directory* option, which defines the destination directory for class files. This value is defined in the "`build.destination`" property located in the tool.config file.
$MAINCLASS$	Contains the name of a class in the project, defining the "`public static void main (String[])`" method. If there is no such class, this macro returns "$MAINCLSS$" and displays the error message in the Messages tab "No class with main() method found".
$LINENUMBER$	Contains the line number of the selection in the file containing the selected element. For example, for selected operation this macro will contain the line in the file with the operation's class.
$CLASS_NAME$	Contains the name of the selected class (fully qualified name in Java).
$PROJECT_DIR$	Contains the full path to the project's directory. For example, if the project is located in the `c:\together\myprojects\CoolProject` directory, then this macro will contain the value `"c:\together\myprojects\CoolProject"`
$PROJECT_NAME$	Contains the name (without extension) of the project file. For example, if the project file is `c:\together\myprojects\CoolProject\myproj.tpr`, then this macro will contain `"myproj"`.
$PROJECT_EXT$	Contains the extension of the project file. For example, if the project file is `c:\together\myprojects\CoolProject\myproj.tpr`, then this macro will contain `"tpr"`.
$PROJECT_FULLNAME$	Contains the name (with extension) of the project file. For example, if the project file is `c:\together\myprojects\CoolProject\myproj.tpr`, then this macro will contain `"myproj.tpr "`.

Table 54 System macros (continued)

$PROJECT_SPEC$	Contains the full name of the project file (path, name and extension). For example, if the project file is c:\together\myprojects\CoolProject\myproj.tpr, then this macro will contain the value: "c:\together\myprojects\CoolProject\myproj.tpr".
$FILELIST$	Contains the "@somefile", where somefile is the name of the automatically generated file with the list of the files (each file on a new line) contained in the selection. (If a user clicks on a class, somefile will contain only the file with the selected class, but if a diagram is clicked, somefile will contain all the files that represent classes in the diagram, and all the classes in subpackages shown on this diagram, etc.)
$FILE _DIR$	Contains the full path to the selected file's directory. For example, if the selected file is located in the c:\together\myprojects\CoolProject directory, then this macro will contain "c:\together\myprojects\CoolProject"
$FILE_NAME$	Contains the name (without extension) of the selected file. For example, if the selected file is c:\together\myprojects\CoolProject\MyClass.java, then this macro will contain "MyClass".
$FILE_EXT$	Contains the extension of the selected file. For example, if the selected file is c:\together\myprojects\CoolProject\MyClass.java, then this macro will contain "java".
$FILE_FULLNAME$	Contains the name (with extension) of the selected file. For example, if the selected file is c:\together\myprojects\CoolProject\MyClass.java, then this macro will contain "MyClass.java".
$FILE_SPEC$	Contains the full name of the selected file (path, name and extension). For example, if the selected file is c:\together\myprojects\CoolProject\MyClass.java, then this macro will contain "c:\together\myprojects\CoolProject\MyClass.java".
DEF_FILE_DIR	Contains the full path to the selected definition file's directory (C++ only). For example, if the selected definition file is located in the c:\together\myprojects\CoolProject directory, then this macro will contain "c:\together\myprojects\CoolProject". Definition files has .cpp extension
DEF_FILE_NAME	Contains the name (without extension) of the selected definition file (C++ only). For example, if the selected definition file is c:\together\myprojects\CoolProject\guest.cpp, then this macro will contain "guest".
DEF_FILE_EXT	Contains the extension of the selected definition file (C++ only). For example, if the selected definition file is c:\together\myprojects\CoolProject\guest.cpp, then this macro will contain "cpp".

Table 54 System macros (continued)

`$DEF_FILE_FULLNAME$`	Contains the name (with extension) of the selected definition file. For example, if the selected definition file is `c:\together\myprojects\CoolProject\guest.cpp`, then this macro will contain `"guest.cpp"`.
`$DEF_FILE_SPEC$`	Contains the full name of the selected definition file (path, name and extension) (C++ only). For example, if the selected file is `c:\together\myprojects\CoolProject\guest.cpp`, then this macro will contain `" c:\together\myprojects\CoolProject\guest.cpp"`.

`$PROMPT$`	Displays a dialog with a text input field and, when you press OK, returns the entered value. If you press CANCEL, the command is cancelled.
	Variants:
	• `$PROMPT:name=placeYourLabelHere$`
	Same as `$PROMPT$`, but you can specify the name (a string after the equal sign) of the label above the input field.
	• `$PROMPT:name=placeYourLabelHere,default=placeDefaultValueHere$`
	Same as above, but allows you specify the default value of the input field. If you press CANCEL, the default value will be returned.
	Note: If there are several $PROMPT$ macros, then only one dialog will be displayed, containing fields specified in these macros, for example:
	`$PROMPT$$PROMPT:name=MyLabel$_someString_$PROMPT$`
	will display a dialog with three input fields, and after you press OK, a string will return containing :
	`AB_someString_C, where A,B,C are the entered values.`

Referencing configuration properties as macros

Besides using the predefined macros listed in Table 54, you can also use any of the properties defined in the `*.config` files. In this case, you need to write `$:nameoftheproperty$`. Note that you must use a colon after the first dollar sign ($). If you use a property named `build.someproperty`, you can write just `$:someproperty$`.

Following are examples:

- `$:build.classpath$` - include the value of the property build.classpath (the value of this property is in the Tools tab (`"Classpath"`) of the Options dialog)
- `$:classpath$` - you can use short names for properties build.*, same as above.

Template Macros

The following macros, surrounded by % characters, are used in the template values for both forward (code generation) and reverse (parser) engineering.

Table 55 Template macros

Macro	Definition	Used in	Language
%Class_Name%	The Choose Pattern dialog does not include this macro. For classes, it is substituted with the name of the class to which this template is applied (*such as* for class constructors). For members, it is substituted with the name of the class where a member is created by this template.	Source-generating templates	Java
%Date_Created%	This macro is expanded into the date of creation in the format: Month XX, YYYY	Prologue and epilogue	Java
%Time_Created%	This macro is expanded into the time of creation in the format: hh:mm:ss AM/PM	Prologue and epilogue	Java
%FILE_NAME%	Name of the class source code file.	Prologue and epilogue	Java
%FILE_EXT%	Extension of the class source code file.	Prologue and epilogue	Java
%Name%	Name of a generated class / attribute / operation, editable in the Choose Pattern dialog.	Source-generating templates	Java
%Dst%	Name of the destination class of a generated link, editable in the Choose Pattern dialog.	Source-generating templates	Java
%Type%	Type of attribute or return type of operation, editable in the Choose Pattern dialog.	Source-generating templates	Java
%Any%	Matches any token.	Source-generating templates	Java

When working with the template macros, you need to consider the following:

- The case of letters in the file macros (%FILE_NAME%, etc.) controls the case of letters in the generated file name. For example, %FILE_NAME%_%FILE_EXT% is expanded to CLASS1_JAVA, while %File_Name%_%file_ext% is expanded to Class1_java.
- The file macros (*%FILE_NAME%*, etc.) are resolved meaningfully. Thus, if due to the context they should be resolved to valid identifiers, you should control this on your own.
- Code templates replace Blueprints from the pre-4.0 versions. If you are still using Blueprints, it is strongly recommended to convert to code templates, as they are more efficient.

Keyboard shortcuts

This section includes keyboard shortcuts, grouped by their location in the Main Menu, and by their functions. For this reason, some of the shortcuts are duplicated in various tables.

- Main window and main menu shortcuts
- Diagrams
- Miscellaneous shortcuts

Main Window and Main Menu

Together provides a keyboard interface for frequently needed tasks. The standard keystrokes are documented in the following tables. The files `action.config` and `menu.config` in your installation store the data that controls the presentation and actions of the keyboard interface. It is possible to modify these files to customize the keyboard interface. However, such customization is recommended for advanced users only.

Note Attention, MacOS users! Though normally in MacOS the Command key is used for the keyboard shortcuts, in Together CTRL key is used instead.

Table 56 Main Window shortcuts

Switching between the inner tabs	ALT+ RIGHT/LEFT ARROW
Expand node	Right arrow, ENTER, double click
Collapse node	Left arrow, ENTER, double click
Next Pane	CTRL+F12
Previous Pane	CTRL+SHIFT+F12
Next Tab	ALT+RIGHT
Previous Tab	ALT+LEFT

Table 57 File shortcuts

New project	CTRL+SHIFT+N
New (Object Gallery)	CTRL+N
Open	CTRL+O
Save	CTRL+S
Save all	CTRL+SHIFT+S
Print Diagram	CTRL+P
Print Documentation	CTRL+SHIFT+P
Close	CTRL+W
Close all	CTRL+SHIFT+W
Exit	ALT+X

Table 58 Edit shortcuts

Undo	CTRL+Z, ALT+BACK_SPACE
Redo	CTRL+Y, CTRL+SHIFT+Z
Cut	CTRL+X, SHIFT+DELETE
Copy	CTRL+C, CTRL+INSERT
Paste	CTRL+V, SHIFT+INSERT
Delete	DELETE
Select All	CTRL+A

Table 59 Search shortcuts

Find	CTRL+F
Replace	CTRL+H
Find Next	F3
Find Previous	SHIFT+F3
Search on Diagrams	CTRL+SHIFT+D
Search / Replace in Files	CTRL+SHIFT+F
Search by Query	CTRL+SHIFT+Q
Search for Usages	CTRL+SHIFT+U

Table 60 View shortcuts

Toggle Explorer Pane	CTRL+ALT+Z
Toggle Designer Pane	CTRL+ALT+D
Toggle Message Pane	CTRL+ALT+M
Toggle Inspector	CTRL+ALT+I
Full screen view	F12

Table 61 Selection shortcuts

New attribute	INSERT, CTRL+A
New operation	INSERT, CTRL+O
New Property	INSERT, CTRL+B
Open Inspector	ALT+ENTER
Rename	F2
Choose Pattern	CTRL+R
Clone	CTRL+SHIFT+C
Delete	Delete
Refactoring/Rename	SHIFT+F2

Table 62 Help shortcuts

Contents	SHIFT+F1

Designer shortcuts

Table 63 Designer shortcuts

Deselect element	ESC
Add Shortcut	CTRL+SHIFT+A
Select all nodes	CTRL+A
Invoke right-click menu of the selected element, when focus is lost	SHIFT+Right Click
Auto layout all elements	CTRL+K
Update	F5
Paste shortcut	CTRL+SHIFT+V
Navigation between elements	UP, DOWN, LEFT, RIGHT
Select first member in the selected class	PAGE DOWN
When member is selected, select its class	PAGE UP
Diagram scrolling up/down/left/right	CTRL+UP, CTRL+DOWN, CTRL+LEFT, CTRL+RIGHT
Diagram scrolling page up/down	CTRL+PAGE UP, CTRL+PAGE DOWN
Diagram scrolling page left/right	CTRL+HOME, CTRL+END
Open Inspector	ALT+ENTER, ALT+double click
Move focus from docked inspector	ALT+ENTER
Delete	DELETE
Rename	F2

Table 64 Zoom shortcuts

Add Shortcut	CTRL+SHIFT+A
Select all nodes	CTRL+A
Activate Zoom Lens	CTRL+SPACE
Zoom in	+
Zoom in with the toolbar zoom icon	click
Zoom out	-
Zoom out with the toolbar zoom icon	ALT + click
Fit in Window	*
Zoom 1:1	/

Table 65 Class diagram shortcuts

New class	CTRL+L
New interface	CTRL+SHIFT+L
New package	CTRL+E

Table 66 Class shortcuts

New attribute	CTRL+A
New operation	CTRL+O
New member by pattern	CTRL+T
New property	CTRL+B

Table 67 Package diagram shortcuts

New class	CTRL+L
New interface	CTRL+SHIFT+L
New package	CTRL+E

Table 68 Attribute, Operation, Property, Statechart, and Activity shortcuts

New attribute	INSERT, CTRL+A
New Operation	INSERT, CTRL+O
New Property	INSERT, CTRL+B
New internal statechart transition	CTRL+T
New internal activity transition	CTRL+T

Miscellaneous

Table 69 Chooser shortcuts

Invoke the right-click menu in the file chooser dialog	SHIFT+F10
Expand node	Numeric keypad plus (Gray +)/ Enter
Collapse node	Numeric keypad minus (Gray -)/ Enter
Open any chooser dialog in the inspector or Options dialog	ALT+INSERT

Table 71 Navigation shortcuts

Switching between open panes	CTRL+F12
Switching between open panes back	CTRL+SHIFT+F12
Switching between the tabs of the selected component	ALT+ RIGHT/LEFT ARROW
Open next message (that can open)	ALT+F10

Table 71 Navigation shortcuts (continued)

Open previous message (that can open)	ALT+F9

INDEX

A

activity diagram 105, 159
  elements 159
  shortcuts 502
  techniques for 161
`addToCompartment` function 452
API
  documentation 480
  IDE 462
  overview 461
  RWI 462
  SCI 462
applet 366
  creating 366
  deploying 366
application client 387
  creating "one-click" 390
application client diagram 303
  deploying 391
  security support 304
  using 391
  visual design elements 390
Application server
  BEA WebLogic 408
  IBM WebSphere 408
  Sun EE reference implementation 407, 408
attribute 468
audits and metrics 273
  additional sources 295
  automatic correction of audit violations 286
  customizing and extending 294
  metrics for audit violations 276
  output for 277
  printing results 285
  running from command line 292
  running in Together 275
  saving results 278
  See also metrics
  sets of, creating and using 287
  sorting results 277
authentication and authorization
  See security

B

bar graph (metrics results) 283
business process diagram 105
  elements 168
  notation 168

C

Choose Pattern command 269
Choose Pattern dialog 256, 263, 268
class diagram 105, 140
  attributes and operations, rearranging 147
  compartment controls, setting 147
  elements 141
  inner class, defining 144
  members and properties, editing 146
  search paths, showing classes of 144
  shortcuts 502
class diagrams
  JavaBeans, showing 144
  links, creating 145
  physical and logical 142
class shortcuts 502
CMP
  EJB 351, 363
Code generation 87
code template
  editing 259
  properties 256
code templates 255
  browsing code 257
  browsing using Directory tab 257
  custom 259
  custom, creating manually 261
  custom, creating using the expert 260
  default properties 257
  displaying custom names 262
  editing 259
  grouping 262
  macros, user-defined 263
  patterns, using with 268
  properties of 256
  source code, formatting 259

using macros 257
collaboration diagram 147
elements 149
messages 152
sequence diagram, converting to 150
command line
parameters 489
syntax 489
component diagram 105, 162
elements 162
techniques for 163
components 36
`config` file 445, 447
referencing configuration properties as macros 497
sample 452
configuration files 445
configuring
overview of levels 56
See options
container transaction 347
creating diagram element 348
diagram element 348
linking to a method 349
container transaction elements
definition 347
container-managed relationship 329
customizing Together
advanced 445

D

database modeling 92
activating in Together 92
declarative security role 434
`def` file 472
default level 56
deployment
view in modeling 104
deployment descriptor 302, 303
encoding 430
limitations 391
resource adapter diagram 397
deployment diagram 105, 164
elements 164
techniques for 165
deployment expert
resource adapter diagram 398
design
view in modeling 104
design elements, standalone 131
diagram level 56
diagrams 127
annotating 130
archived file, restore from 88
autodependency links, filtering out 116
bidirectional links, converting 119
cloning 109
configuring options 109
creating custom types 446
creating using main menu 108
default of a project 74
dependency links 115
drag-and-drop support 118
editing properties 124
element size 112
elements and links 117
elements, copying and cloning 118
elements, drawing 110, 111
elements, moving and copying 118
elements, resizing 119
elements, selecting 117
graphics, improving 110
grid, using the 111

hyperlinking 127
hyperlinking feature 108
images 132
layout 121
layout, automated 122
layout, manual 123
link to self, drawing 115
links, drawing 114, 115
links, labeling 120
list of supported 104
multiple elements, adding 113
opening and closing 123
overview for creating 107
printing 133
renaming 109
right-click menus 117
See also J2EE-supported diagrams
See also UML extension diagrams
See also UML modeling
See specific types of diagrams
undo/redo 111
zooming view 122
distribution graph (metrics results) 284
DocGen
See documentation generator
documentation generation
automating 182, 492
options 184
RTF options 178
starting 173
documentation generator 176, 187
documentation template 176, 187
body sections 188
calls to stock sections 189, 200
calls to template sections 189, 201, 219
controls 203
current model element 189
data controls 207
designer 187
DG functions 230
DG variables 227, 228
element iterator scopes 197
element iterators 189, 196
element property iterators 189, 198
enabling condition 195
folder sections 189, 199
footers 203
formula controls 209
frameset templates 215
headers 203
hypertext links 212, 221
image mapping 222
images 205
JavaDoc link references 223
labels 205
metamodel 190
panels 205
root object metatype 189
static sections 189, 203
stock sections 200
text controls 210
variables 209
documentation templates
DG functions 230
drag-and-drop support 118

E

early access folder 467
editing
code templates 259
inplace, how to enable 452
EJB 307
assembler diagram relates to J2EE deployment expert, how 344
component synchronization options 309
configuring using EJB inspectors 315
Copy options 309
correction 343
creating a project using existing EJB code 315
creating using the Object Gallery dialog 314
entity 312
message-driven 312
resource adapter 393
session 312
shortcuts, creating 340
show options 309
verification 343
EJB assembler diagram 302
and J2EE Deployment Expert 344

creating 338
display 341
properties 339
security support 304
toolbar elements 339
EJB assembler diagrams 337
EJB components
class diagram toolbar buttons 312
creating 307
customizing default code 335
editing 309
entity beans 327
Inspectors 315
local interfaces 312
message-driven beans 332
methods 321
Object Gallery 312
properties 307
references 317
session beans 330
sharing interfaces 333
EJB deployment
properties 318
EJB modules 337
creating 338
creating a project using existing EJB code 315
creating EJBs 308
security support 433
visual design elements 312, 339
EJB modules
See also EJB 337
EJB pattern 313
EJB properties
diagram element 350, 362
using 351, 363
EJB Properties element 351
EJB reference
and reference attribute 354
creating diagram elements 353
diagram element 350
EJB resource-environment reference, creating 356
EJB reference element 350
EJB references
creating 351
environment references, creating 356
environment variable, changing type 357
resource references, creating 355
supported types 352
EJB shortcuts 340
element
options, defining 451
shape of, defining 451
unique name 451
element, Together 25
encoding 430
enterprise application
creating using Object Gallery 380, 388
security support 440
enterprise application diagram 302
and J2EE deployment expert 384
creating 380, 388
elements 381
importing files 383
shortcuts, creating 381
enterprise applications 379
visual assembly 379
entity bean
See entity EJB 327
entity beans
default code 327
fields 320
importing from database 313

persistence 316, 328
primary key 317, 329
entity EJB 312
code template 258
references 317
Explorer pane 30
Components tab 36
Copying and pasting 37
Diagrams tab 34
Directory tab 30
Favorites tab 33
finding locations 38
Model tab 31
Modules tab 35
opening diagrams and files 32
quick project access 37
Server tab 31
toolbars 33
using right-click menus 37
export 91
database 93
DDL 93
IDL 95
JDBC supported types 92
XMI 97
extension 264
codegen default 264
custom 264
source 264

F
filter mappings 373
filters 373
creating 373
deploying 375
focus pane 25

G
generating documentation
See documentation generator
getter 468
graphics
improving in diagrams 110

H
Hello World example 464
hyperlinking 127
browsing 129
diagrams and elements 127

I
IDE 461, 462
IdeScript interface 463
IdeStartup interface 463
implementation
view in modeling 104
import 91
database 94
existing files to an enterprise application diagram 383
JDBC supported types 92
XMI 96
Inspector
EJB component 315
inspector 25, 40
EJB 315
web application diagram 381, 390
instances, running multiple 82
Integrated Development Environment
See IDE
interface
sharing home and remote 333
interface, definition of 463

J
J2EE Connector Architecture 393
J2EE deployment 399
Command Line File 417
fast track 406
full featured 406
required installations 401
transitioning among application servers 426
J2EE Deployment Expert 344, 400
BEA WebLogic Run-Time Deploy 419
common server options 409
config files 406
deployment diagrams 404
IBM WebSphere 3.5 421
IBM WebSphere 4.0 425

process options 409
simple JSP client 415
Starting application servers 402
Sun EE Deployment 418
supported application servers 400
Verify / Correct Sources 414
J2EE Module Import command 88
J2EE support 299
See also references support
J2EE-supported diagrams 300
application client diagram 303
EJB assembler diagram 302
enterprise application diagram 302
resource adapter diagram 303
Taglib diagram 303
web application diagram 302
JAR 387
JSP 369
creating new 369

K
keyboard shortcuts 499
designer 501
main menu 499
Kiviat graph (metrics results) 282

L
launcher
`Together.exe`, invoking 493
listeners 375
and Web applications 376
creating 376
event types 375
LiveSource 85, 95, 96

M
macros
for code templates 263
system 494
template 498
main
menu 28
window 25
main class, Together 491
`manifest` file
See `Manifest.mf` file, `def` file
`Manifest.mf` file 472
message-driven beans
default code 332
deployment properties 333
project properties 332
message-driven EJB 312
method 468
metrics
comparing results 280
graphs 282
reports 279
saving and loading results 280
See also audits and metrics
model
existing code, create from 85
modeling
deployment view 104
design view 104
implementation view 104
list of supported diagrams 105
overview 103
patterns 104
process view 104
See also diagrams
See UML modeling
use case view 104
module
definition 462
modules 35
activating and deactivating 55

compiling 477
declaring 472
declaring in `def` 475
declaring in Manifest.mf 472
evaluating results 478
extending Together 465
guidelines for developing 467
Hello World example 464
interfaces implemented by 466
locating 463
naming conventions 468
registering 464
running 467
running at start-up 493
source code 470
starting 464
storing compiled class 477
tab, description of contents 467
troubleshooting 478
tutorial for writing and deploying 470
types of 466
viewing 466

N

New Project dialog 75

O

Object Gallery
EJB, creating a 312
options
checkboxes, value of 61
context-sensitive help 61
reference guide 61
resizing the Options dialog 61
viewing and editing 60
Options dialog 459

P

package diagram
shortcuts 502
package diagrams 142
working with 143
pane 25
expand 27
hiding title bars 27
resize 26
undocking 27
view or hide 25
`parameter` function 451
parameters
command line 489
Together main class 491
virtual machine preferences 493
Parser blueprints 88
defining 88
pattern
resource adapter 395
patterns 265
and modeling 104
behavior 266
choosing for member 271
classes and links, creating 269
code templates, using with 268
creating 266
creating members by 270
descriptions, providing 268
elements, generated 266
refactoring with 272
performance tuning
large projects 84
process
view in modeling 104
programmatic security role 434
project
creating a 75
creating using existing EJB code 315
default diagram 74
multiple instances, running 82
New Project dialog, using 75
performance tuning 84
properties, editing 76
resource options, setting 81
resources, adding 79
resources, removing 81
root directory, primary 74
tips for large projects 83
`tpr` file 74
understanding basics 73
views with referenced content 84
project level 56

project properties
application server 307
`properties` file 446

Q
QA module 294
extending 294
See also audits and metrics

R
Read-Write Interface
See RWI
refactoring
encapsulating attributes 251
extracting interfaces, superclasses 252
move class, interface 248
patterns 272
pull up operation, attribute, member 248
push down operation, attribute, member 248
renaming 250
Reference attribute
Inspector 352
reference attribute 350
in EJB client 351
types 352
references support
security support 304
resource adapter
connection classes 395
Connector Architecture 393
resource adapter diagram 303
creating 394
creating by a pattern 395
elements 394
security support 304
using 397
resource adapters 393
resource files 446
resources
adding to a project 79
removing from a project 81
setting options 81
reverse engineering 85
J2EE archive files 88
J2EE Module Import command 88
right-click menu 25
in diagrams 117
robustness analysis diagram 168
elements 169
key elements and properties 170
root directory
of a project 74
RWI 462
*See*Read-Write Interface
RWI properties 190

S
sample folder 466
SCI 461, 462
script, definition of 463
Security 431
method permission 433
EJB modules 433
enterprise applications 440
method permissions 434
principals 433
resource adapters 440
resource authentication mechanisms 441
security constraints 438
security role 432
security role reference 435
security role references 433

Web applications 437
Web resource collection 438
security
authentication and authorization 438
declarative security role 434
EJB modules 433
enterprise application 440
J2EE platform support 431
programmatic security role 434
roles 432
Web applications, in 437
security role 437
See keyboard shortcuts 499
sequence diagram 105, 147
collaboration diagram, converting to 150
elements 149
generating from operation 150
lifelines 155
messages 152
Options dialog 151
source code, generating 156
Server Explorer
starting servers from 402
servlet 366
creating 366
creating using Object Gallery 368
Web application 369
servlets
creating using pattern 367
session bean
See session EJB
session beans
default code 330
stateful and stateless 331
session EJB 312, 330
`setGraphicObject` function 451
`setLayoutConstraints` function 451
setter 468
shortcuts
creating 116
EJB, creating for 340
in the Explorer 34
Show options
adding 459
display text, changing 458
removing 459
Source Code Interface
See Source Code Interface
startup module 475
statechart diagram 105, 156
elements 157
entry/exit actions, specifying 158
internal transition, creating 157
self-transitions, drawing 157
shortcuts 502
substates, nested 158
transition sources and targets, showing multiple 158
stereotype
UML diagram support of 138
system folder 466
system macros 494

T

tag handler 370
properties 372
tag library
See TagLib
TagLib 370
creating 372
TagLib diagram 371
elements 371
properties 371
taglib diagram 303
template
default properties 257
using macros 257
template macros 498
templates
See code templates
Together diagrams
See UML Extension Diagrams
`Together.exe` launcher 493
`Together_starter` command 490
`tpr` file 74
transaction attribute
See container transaction
troubleshooting
modules 478

U
UML extension diagrams 167
business process diagram 167
robustness analysis diagram 168
UML modeling
activity diagram 159
class diagram 140
collaboration diagram 147
component diagram 162
deployment diagram 164
list of supported diagrams 137
notation in diagrams 137
sequence diagram 147
statechart diagram 156
stereotypes 137
use case diagram 138
use case
view in modeling 104
use case diagram 105, 138
browse-through sequences 139
elements 138
extension points 140
user interface
advanced customization 445
diagram configuration file 447
icons and icon references, creating 447
toolbar icons, defining 448
treeview icons, defining 448
viewmaps, defining 450
Using 315

V
view management
customizing Show options 458
virtual machine 493

W
Web application
security 438
security support 437
Web application diagram 359
and J2EE Deployment Expert 377
creating 360
elements 361
properties 362
shortcuts 365
web application diagram 302
security support 304
web applications 359
WebLogic Inspector tabs 324
CMP entity EJBs 325
entity EJBs 325
message-driven EJBs 327
session EJBs 323, 324
WebLogic 6.0 /6.1 324
WebLogic 7.0 324

X
XML 397

Z
zoom shortcuts 501